Master Thesis

Modeling a Xenon Tank, its Heat Convection and In-Orbit Behavior with a 2D CFD Method

Alexander Merkel

Diese Arbeit wurde vorgelegt am
Institut für Strahlantriebe und Turbomaschinen

Fakultät für Maschinenwesen der
Rheinisch-Westfälischen Technischen Hochschule Aachen

Erster Prüfer:	Univ.-Prof. Dr.-Ing. Michael Oschwald
Betreuer Intern:	Jan Bisping, M.Sc.
Betreuer OHB:	Bayrem Zitouni

Aachen, December 2019

Abstract

Xenon becomes increasingly important as a propellant for spacecrafts. As part of the development process, the thermal control of the tank during the loading and unloading procedure has to be designed. To aid in this task, a 2D model of xenon inside its tank during the pressurization and depressurization process, specifically for a zero-g environment, has been created using OpenFOAM. The physical theory behind fluid flows as well as the methodology the model creation is based upon and the used numerical settings are presented. The model was applied to compute the fluid properties and heat flow during xenon loading and unloading for the *ESPRIT* project. When comparing the results to 0D/1D simulations performed by EcosimPro and an internal Excel tool, phenomena not modeled with these approaches could be observed in the OpenFOAM results. This makes it a valuable addition to the tools available during the design of the propulsion system.

Zusammenfassung

Xenon gewinnt stetig an Bedeutung als Treibstoff für Raumfahrzeuge. Um dies zu ermöglichen muss ein Thermalsystem ausgelegt werden, welches den Tank während des Be- und Entladevorgangs in seinem festgelegten Temperaturbereich hält. Zur Unterstützung der Auslegung wurde ein 2D Modell mittels OpenFOAM erstellt, welches den Be- und Enttankungsvorgang, ins besondere in Schwerelosigkeit, modelliert. Die Theorie zur Strömungsdynamik sowie die Methodik zur Modellierung und die verwendeten numerischen Verfahren werden vorgestellt. Das Modell wurde auf den Be- und Entladevorgang für das *ESPRIT* Projekt angewendet um die Fluideigenschaften und den Wärmestrom zu berechnen. Im Vergleich der Ergebnisse mit Simulations, die mit EcosimPro und einem internen Excel Tool durchgeführt wurden, konnte gezeigt werden, dass die 2D Simulation Phänomene darstellt, die mittels der anderen Methoden nicht abgebildet werden können. Dies macht das erstellte 2D Modell zu einer wertvollen Ergänzung der Werkzeuge, die bei der Entwicklung eines Antriebssystems zur Verfügung stehen.

Contents

Nomenclature **v**

Abbreviations **viii**

1. Introduction **1**
1.1. Electric Propulsion 1
1.2. ESPRIT 3
1.3. Xenon tank 4
1.4. EcosimPro 4
1.5. Excel Tool 5
1.6. Objective of this Thesis 5

2. Theory of Fluid Flows **6**
2.1. Continuity Equation 7
2.2. Momentum Equation 8
2.2.1. Surface forces 9
2.2.2. Body forces 9
2.2.3. Stress tensor 10
2.3. Energy Equation 10
2.4. Dimensionless Quantities 13
2.4.1. Reynolds number 13
2.4.2. Prandtl number 14
2.4.3. Grashof number 15
2.4.4. Nusselt number 15
2.4.5. Rayleigh number 15

3. Computational Fluid Dynamics **16**
3.1. OpenFOAM 16
3.2. Solver selection 16
3.3. rhoPimpleFoam 18
3.3.1. SIMPLE 18
3.3.2. SIMPLEC 21
3.3.3. PISO 21
3.3.4. PIMPLE 23

3.4. Numerical Schemes . . . 24
3.4.1. Temporal Discretization . . . 25
3.4.1.1. Forward Euler Scheme . . . 25
3.4.1.2. Backward Euler Scheme . . . 25
3.4.1.3. Crank-Nicolson Scheme . . . 26
3.4.2. Convection Discretization . . . 26
3.4.2.1. Upwind scheme . . . 27
3.4.2.2. Central Difference scheme . . . 27
3.4.2.3. Second Order Upwind scheme . . . 28
3.5. Turbulence Modeling . . . 29
3.5.1. Standard $k-\varepsilon$ Model . . . 30

4. Tank model **32**
4.1. General Structure . . . 32
4.2. Mesh and Solver Setup . . . 33
4.2.1. Mesh . . . 33
4.2.2. Schemes and Solution Algorithm Control . . . 35
4.3. Thermophysical and Turbulence Properties . . . 36
4.3.1. Thermophysical Properties . . . 36
4.3.2. Turbulence Properties . . . 37
4.4. Initial and Boundary Conditions . . . 37
4.4.1. Initial conditions . . . 37
4.4.2. Boundary conditions . . . 37
4.4.2.1. Inlet/Outlet port . . . 38
4.4.2.1.1. Pressure . . . 38
4.4.2.1.2. Temperature . . . 38
4.4.2.1.3. Velocity . . . 38
4.4.2.2. Wall . . . 39
4.4.2.2.1. Pressure . . . 39
4.4.2.2.2. Temperature . . . 39
4.4.2.2.3. Velocity . . . 39
4.4.2.2.4. Turbulence . . . 39
4.5. Comparison models . . . 40
4.5.1. Excel tool . . . 40
4.5.2. EcosimPro Model . . . 42

5. Results **44**
5.1. Parameter study . . . 44
5.1.1. Cell size . . . 44
5.1.2. Time step . . . 46
5.1.3. Processor cores . . . 47

5.2. Client case 49
5.2.1. EcosimPro comparison 55
5.2.2. Excel tool comparison 56
5.3. Servicer case 57
5.3.1. EcosimPro comparison 62

6. Conclusion **64**
6.1. Summary 64
6.2. Future Work 66

List of Figures **67**

List of Tables **69**

Bibliography **70**

A. Usage guide for Python scripts **74**
A.1. nistToOpenFoam.py 74
A.2. meshGen.py 75

Nomenclature

c_P	Specific heat capacity at constant pressure	J/(kgK)
e	Specific total energy	$\frac{\text{J}}{\text{kg}}$
$\tilde{e}$	Normalized error	-
E	Total energy	J
$\vec{f}$	External forces	N
$\vec{f}_S$	Surface forces	N
$\vec{f}_b$	Body forces	N
F_T	Thrust	N
Gr	Grashof number	-
$\vec{g}$	Gravitational acceleration	m/s^2
g_0	Standard acceleration due to gravity	m/s^2
h	Specific enthalpy	$\frac{\text{J}}{\text{kg}}$
h	Heat transfer coefficient	$\text{W/(m}^2\text{K)}$
I_{sp}	Specific impulse	s
$\mathbf{I}$	Identity tensor	-
k	Thermal conductivity	W/(mK)
k	Turbulence kinetic energy	W/(Jkg)
L	Characteristic length	m
m	Mass	kg
m_P	Propellant mass	kg
m_0	Initial mass	kg
$\dot{m}$	Mass flow	$\frac{\text{kg}}{\text{s}}$

M	Molar mass	$\frac{\text{kg}}{\text{mol}}$
$\vec{n}$	Normal vector of a surface	-
Nu	Nusselt number	-
p	Pressure	Pa
p_e	Exhaust pressure	Pa
p_0	Ambient pressure	Pa
Pr	Prandtl number	-
$\dot{Q}$	Heat flow	W
$\dot{Q}_S$	Heat flow across surfaces	W
$\dot{Q}_V$	Heat changes inside the volume	W
R	Ideal gas constant	$\text{m}^3\text{Pa}/(\text{molK})$
Ra	Rayleigh number	-
Re	Reynolds number	-
S	Volume surface	m^2
$\tilde{t}$	Normalized computation time	-
T	Temperature	K
T_s	Surface temperature	K
T_∞	Ambient temperature	K
$\mathbf{T}$	Total stress tensor	N/m^2
U	Velocity	$\frac{\text{m}}{\text{s}}$
$\dot{U}$	Change of internal energy	$\frac{\text{J}}{\text{s}}$
u	Velocity in x direction	$\frac{\text{m}}{\text{s}}$
v	Velocity	$\frac{\text{m}}{\text{s}}$
v	Velocity in y direction	$\frac{\text{m}}{\text{s}}$
$\vec{v}$	Velocity	$\frac{\text{m}}{\text{s}}$
v_e	Exhaust velocity	$\frac{\text{m}}{\text{s}}$
V	Volume	m^3
w	Velocity in z direction	$\frac{\text{m}}{\text{s}}$
$\dot{W}$	Technical work	W
$\dot{W}_S$	Work performed by surface forces	W
$\dot{W}_b$	Work performed by body forces	W
$\vec{x}$	Position vector	m
α	Thermal diffusivity	m^2/s
α_t	Turbulent thermal diffusivity	m^2/s

α	Phase volume fraction	-
β	Coefficient of thermal expansion	$\frac{1}{\mathrm{K}}$
Γ	Source term	-
Δv	Velocity increase	$\frac{\mathrm{m}}{\mathrm{s}}$
ϵ	Specific internal energy	$\frac{\mathrm{J}}{\mathrm{kg}}$
ε	Turbulence energy dissipation rate	J/(kgs)
η	Turbulent micro length	m
μ	Dynamic viscosity	$\mathrm{s/m^2}$
μ_P	Propellant mass fraction	-
ν	Kinematic viscosity	$\mathrm{m^2/s}$
ν_t	Turbulent kinematic viscosity	$\mathrm{m^2/s}$
ρ	Density	$\mathrm{kg/m^3}$
$\overline{\overline{\tau}}$	Viscous stress tensor	$\mathrm{N/m^2}$
ϕ	Fluid property	-
φ	Fluid property	-
Ψ	Crank-Nicolson off-centering coefficient	-

Abbreviations

Abbriviation	Description
CFD	Computational fluid dynamics
CPU	Central processing unit
CSA	Canadian Space Agency
CV	Control volume
DIC	Simplified Diagonal-based Incomplete Cholesky
ESA	European Space Agency
ESPRIT	European System Providing Refueling, Infrastructure and Telecommunications
ESPSS	European Space Propulsion System Simulation
FOAM	Field Operation And Manipulation
GAMG	Geometric agglomerated algebraic multigrid
JAXA	Japan Aerospace Exploration Agency
MV	Material volume
NASA	National Aeronautics and Space Administration
NIST	National Institute of Standards and Technology
OHB	Orbitale Hochtechnologie Bremen
PISO	Pressure-Implicit with Splitting of Operators
SIMPLE	Semi-Implicit Method for Pressure-Linked Equations
SIMPLEC	Semi-Implicit Method for Pressure Linked Equations-Consistent
URL	Uniform Resource Locator

1. Introduction

When increasing the pressure p of a compressible fluid in an adiabatic system, the temperature of the fluid T increases. This can be derived from the ideal gas law

$$pV = \frac{m}{M}RT \tag{1.1}$$

which gives a proportionality between the pressure and the temperature for an ideal gas that depends on the volume V, the mass m and the molar mass of the fluid M. For a fixed volume and mass, the temperature rises faster if the molar mass is bigger.

Xenon becomes increasingly important for space missions as a propellant for electric thrusters. Inside the tanks, the xenon is stored at a very high pressure of more than 100 bar. This results in large thermal loads on the tank and the spacecraft during loading and unloading that have to be considered during the design process and make timely loading of large propellant masses challenging.

A number of researchers have looked at this problem from the on-ground loading perspective using numerical [12, 11] and experimental [7] approaches. For the *ESPRIT* project, the in-orbit behavior of xenon becomes relevant.

In this thesis, a 2D model for a xenon tank in a zero-g environment is described. Initially, the mathematical and physical theory behind fluid flows is presented. Afterwards, the methods of Computational Fluid Dynamics are described and the setup of the numerical model is presented. Finally, the results of the 2D simulations are discussed and compared 1D/0D simulations performed using EcosimPro and an internal Excel tool.

1.1. Electric Propulsion

In order to generate thrust, a space propulsion system exhausts a gas at a high velocity in the opposite direction of the desired acceleration vector. In accordance

with Newton's Third law of motion[15], this results in a thrust force in the aspired direction. In general, the methods of accelerating the exhaust gas can be grouped into three categories: Cold gas systems, chemical systems, and electrical systems. The generated thrust can be calculated using the general thrust equation[28]

$$F_T = \dot{m} v_e + (p_e - p_0) A_e \tag{1.2}$$

where $\dot{m}$ is the exhaust gas mass flow and v_e the exhaust velocity. Generally, the pressure term can be neglected because a convergent-divergent nozzle is used to expand the exhaust gas to the ambient pressure p_0. Hence, it can be said that the thrust depends on the mass flow and the exhaust velocity.

In a cold gas thruster, a pressurized inert gas is expanded through a convergent-divergent nozzle into the vacuum of space, thus generating thrust. This system has the advantage of being cheaper and less complex than the other two systems, however this comes at the expense of thrust and efficiency.[32] In a chemical propulsion system, the chemical potential of the fluid (monopropellant system) or fluids (bipropellant system) is used to increase the pressure and temperature of the exhaust gas dramatically before expanding it using a convergent-divergent nozzle. This method uses very large mass flow compared to the other two, resulting in a large thrust vector. In an electric propulsion system, electric energy is used to accelerate an inert gas to generate thrust. This acceleration can be accomplished by heating the exhaust gas and expanding it through a convergent-divergent nozzle (electrothermal propulsion) or by ionizing the gas and accelerating the ions using an electric (electrostatic propulsion) or an electromagnetic field. As a propellant for an electrostatic thruster, xenon or krypton are typically used.[28, 45] These thrusters have a very small propellant mass flow, but the exhaust gases are accelerated to very high velocities ($> 15\,\frac{\mathrm{km}}{\mathrm{s}}$ for a hall-effect thruster[22]). While the generated thrust is very small compared to a chemical propulsion system, the specific impulse

$$I_{sp} = \frac{v_e}{g_0} \tag{1.3}$$

is large ($> 1500\,\mathrm{s}$ compared to $\approx 320\,\mathrm{s}$ for a typical bipropellant system[28]). Therefore, the propellant mass fraction μ_P, defined as

$$\frac{m_P}{m_0} = \mu_P = 1 - \exp\left(-\frac{\Delta v}{v_e}\right) \tag{1.4}$$

required to achieve a given velocity increase Δv is much smaller for an electric propulsion system compared to a chemical one.

Due to the low thrust, electric propulsion systems are not viable for boosters, but they are increasingly used on satellites as orbit raising and station keeping engines.[10] The *Hispasat 36W-1* satellite built by OHB uses OKB Fakel SPT100 hall-effect thrusters as part of the station keeping system. Additionally, an all electric satellite, meaning electric propulsion for the orbit raising as well as for the station keeping, called *Electra* is currently in development for ESA.[36]

1.2. ESPRIT

The *Lunar Orbital Platform-Gateway* is a NASA-led project involving the ESA, Roscosmos, JAXA and CSA. The goal of the project is to create a space station in the lunar orbit that can serve as a communication hub, a habitation module, and a transfer gateway for lunar missions, specifically the Artemis missions.[30, 31] One of the major european contributions to the *Gateway* is the *European System Providing Refueling, Infrastructure and Telecommunications* module (short *ESPRIT*). This module is currently in the development stage at Thales Alenia Space under contract from ESA. OHB System AG is a subcontractor for two studies involving the structure, the thermal control system and a refueling system.[35] One of the functionalities of the *ESPRIT* module will be refueling the *Gateway*'s xenon tanks from a visiting vehicle.

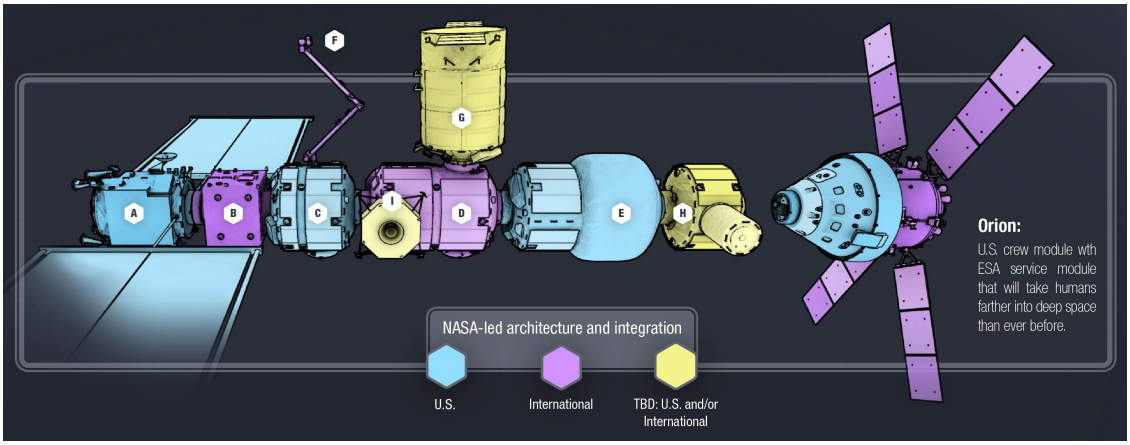


Figure 1.1.: Planned configuration of the *Gateway* with the *ESPRIT* module **B** [31]

1.3. Xenon tank

As previously mentioned, an electric propulsion system typically uses xenon or sometimes krypton as propellant fluid. The tank proposed for *ESPRIT* is Northrop Grumman's 80458-1 Xenon Propellant Tank. This is a composite overwrapped pressure vessel with a center cylinder welded to dome-like end caps and overwrapped with T1000 carbon fiber. The tank is rated for an operational pressure of 186.16 bar, has an internal diameter of 419.1 mm, an internal height of 1125.22 mm, and a propellant volume of 132.74 l. The tank is mounted using polar bosses and is connected via an inlet/outlet tube with a diameter of 9.525 mm.[34]

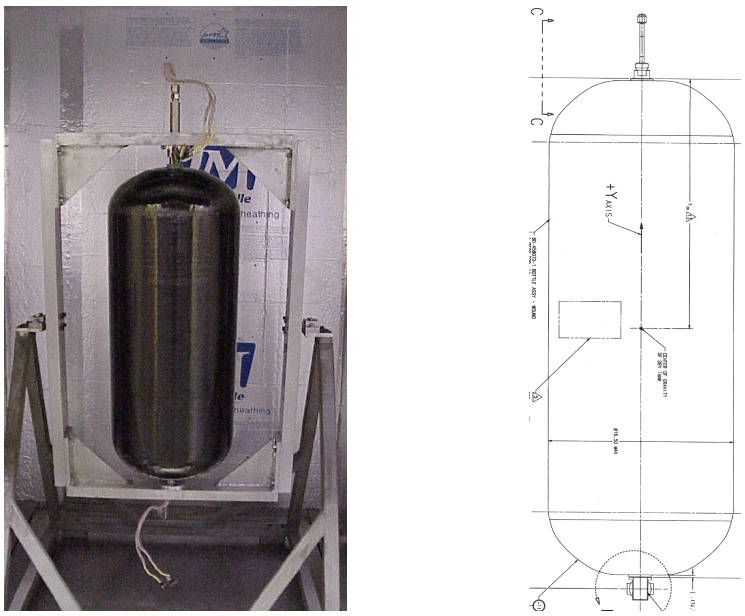


Figure 1.2.: Picture and schematics of the 80458-1 [34]

1.4. EcosimPro

EcosimPro is a simulation software developed by the Spanish company *Empresarios Agrupados* since 1989 with the original funding coming from ESA. It can be used to model physical processes with 0D or 1D equations utilizing a graphical user interface and/or an object-oriented programming language called EL.[8] To speed up and simplify the creation of new models, libraries of components have been developed. Each component contains the modeling equations for the physical objects it represents. One of said libraries is the *European Space Propulsion System Simulation* (short ESPSS) which contain a number of components typically needed

in propulsion systems, such as tanks, valves, pumps, combustion chambers, and nozzles.[25] Using these components, a 1D model of a propulsion system or parts of it can be modeled with relative ease. At OHB, EcosimPro and the ESPSS are used extensively, especially during early development.

1.5. Excel Tool

Internally at OHB, an Excel tool was created to estimate the tank shell temperature during tank pressurization on ground using a 0D method. The assumptions and the used formulas are discussed in chapter 4.5.1.

1.6. Objective of this Thesis

The intention is to create a 2D model that can be used to analyze the xenon tank pressurization and its consequences for the thermal control system in a zero-g environment, and to perform this analysis for the *ESPRIT* case. The model shall be created using the open source framework OpenFOAM. Since OpenFOAM ships with a number of solvers optimized for various fluid dynamics problems, a suitable solver shall be selected and appropriate numerical schemes shall be investigated. A parameter study shall be performed to find suitable settings for the time step, the processor cores and the mesh size. The resulting model shall then be applied to the ESPRIT case and the calculated results shall be compared to results created using EcosimPro and the above mentioned Excel tool.

2. Theory of Fluid Flows

The described problem concerns xenon as a fluid, in either its gaseous or its supercritical state. Therefore, in this chapter the mathematical descriptions of a fluid flow are briefly discussed. There exists a lot of literature that goes into detail about the mathematics and physics beyond what is presented here.[26, 9, 17]

In general, a fluid flow is described by the continuity, momentum and energy equations which describe the conservation of mass, momentum and total energy respectively. These are collectively known as the Navier-Stokes equations and are highly nonlinear second order partial differential equations with four independent variables (three spacial coordinates and one temporal coordinate).

There are two different approaches to describing the conservation laws, namely the Lagrangian and the Eulerian approach. In the Lagrangian approach, the fluid is divided into fluid parcels that are tracked as they move through time and space. Each parcel is tagged by a position vector $\vec{x}_0$, usually set in the parcel's center of mass at time t_0. The movement of the parcel then is described by the function $\vec{x}(t,\vec{x}_0)$. The Eulerian approach on the other hand focuses on a specific volume element through which the fluid flows over time. The flow variables are therefore a function of the position $\vec{x}$, the time t and the flow velocity $\vec{v}(\vec{x},t)$. The two descriptions are related by

$$\vec{v}(\vec{x}(\vec{x}_0,t),t) = \frac{\partial}{\partial t}\vec{x}(\vec{x}_0,t) \tag{2.1}$$

To describe the change of a material volume in the Eulerian specification, Reynolds Transport Theorem is used. It gives a relation between the change of a fluid property ϕ in a material volume over time with the volume integral and the surface fluxes of this property

$$\left(\frac{d\phi}{dt}\right)_{MV} = \int_V \left[\frac{\partial}{\partial t}\left(\rho\frac{d\phi}{dm}\right) + \nabla\cdot\left(\rho\vec{v}\frac{d\phi}{dm}\right)\right] dV \tag{2.2}$$

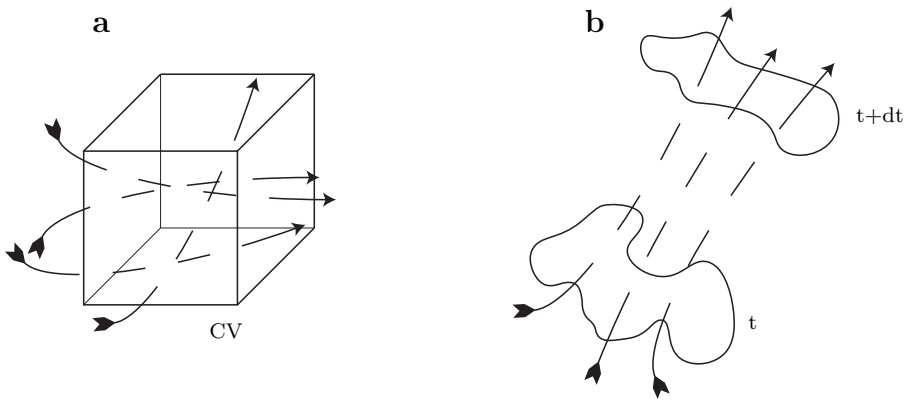


Figure 2.1.: **a** Eulerian and **b** Lagrangian representation of the fluid flow

2.1. Continuity Equation

The continuity equation describes the conservation of mass, meaning without sources or sinks, an object will have a constant mass. In the Lagrangian system this can be written as

$$\left(\frac{dm}{dt}\right)_{MV} = 0 \tag{2.3}$$

We can convert this into the Eulerian system using Eq. (2.2). For a fluid of mass m, density ρ, and velocity $\vec{v}$, this will give

$$\frac{\partial \rho}{\partial t} + \nabla \cdot [\rho \vec{v}] = 0 \tag{2.4}$$

or in the integral form for a control volume V

$$\int_V \left(\frac{\partial \rho}{\partial t} + \nabla \cdot [\rho \vec{v}]\right) dV = 0 \tag{2.5}$$

For an incompressible fluid, the density ρ does not change over time, therefore Eq. (2.4) can be simplified to

$$\nabla \cdot \vec{v} = 0 \tag{2.6}$$

or in the integral form with the control volume surfaces S and their normal vectors $\vec{n}$

$$\oint_S (\vec{v} \cdot \vec{n})\, dS = 0 \tag{2.7}$$

2.2. Momentum Equation

The conservation of momentum describes that without external forces acting on a body, the total momentum of the body will remain constant. Additionally, since the momentum is a vector quantity, the directional components will also be conserved. This phenomenon is described in Newton's Second Law of Motion. For a material volume and in the Lagrangian System, this can be written as

$$\left(\frac{d(m\vec{v})}{dt}\right)_{MV} = \left(\int_V \vec{f} dV\right)_{MV} \tag{2.8}$$

where $\vec{f}$ is the sum of the external forces acting on the material volume with mass m, density ρ, and velocity $\vec{v}$. Using Eq. (2.2), one can write this in the conservative form for an Eulerian system

$$\frac{\partial}{\partial t}\left[\rho\vec{v}\right] + \nabla \cdot (\rho\vec{v}\vec{v}) = \vec{f} \tag{2.9}$$

or in the integral form with a control volume V

$$\int_V \left[\frac{\partial}{\partial t}\left[\rho\vec{v}\right] + \nabla \cdot (\rho\vec{v}\vec{v}) - \vec{f}\right] dV = 0 \tag{2.10}$$

The external forces $\vec{f}$ consist of the surface forces $\vec{f_S}$ and the body forces $\vec{f_b}$ so that

$$\vec{f} = \vec{f_S} + \vec{f_b} \tag{2.11}$$

2.2.1. Surface forces

The forces acting on a volume element result from the pressure and viscous stresses. These can be expressed in terms of the total stress tensor $\mathbf{T}$ where T_{ii} represents the normal stresses and T_{ij} the shear stresses acting on face i in the j direction. In practice, the stress tensor can be split such that

$$\mathbf{T} = -p\mathbf{I} + \overline{\overline{\tau}} \tag{2.12}$$

where $\mathbf{I}$ is the identity tensor, p the pressure and $\overline{\overline{\tau}}$ the viscous stress tensor.

Therefore, the surface forces can be expressed as

$$\vec{f_S} = \nabla \cdot \mathbf{T} = -\nabla p + \nabla \cdot \overline{\overline{\tau}} \tag{2.13}$$

2.2.2. Body forces

There are a number of forces acting on the volume body, such as the Coriolis forces and the Centrifugal forces for a rotating body. However, the predominant body force is the gravitational force. Therefore, the body forces $\vec{f_b}$ can usually be written as

$$\vec{f_b} = \rho\vec{g} \tag{2.14}$$

where $\vec{g}$ is the gravitational acceleration vector.

Using Eq. (2.11), (2.13), and (2.14), the momentum equation (2.9) can be written as

$$\frac{\partial}{\partial t}\left[\rho\vec{v}\right] + \nabla \cdot (\rho\vec{v}\vec{v}) = -\nabla p + \nabla \cdot \overline{\overline{\tau}} + \rho\vec{g} \tag{2.15}$$

2.2.3. Stress tensor

The momentum equation (2.15) contains the viscous stress tensor $\overline{\overline{\tau}}$. It is desirable to express the stresses in terms of flow variables instead. For a Newtonian fluid, i.e. a fluid with a linear relationship between shear stress and shear rate, the shear tensor is given by

$$\overline{\overline{\tau}} = \mu \left(\nabla \vec{v} + (\nabla \vec{v})^{\intercal} - \frac{2}{3} (\nabla \cdot \vec{u}) \mathbf{I} \right) \tag{2.16}$$

where μ is the dynamic viscosity, $^{\intercal}$ indicates the transpose of $\nabla \vec{v}$, and $\mathbf{I}$ is the identity tensor.

Substituting the stress tensor $\overline{\overline{\tau}}$ in Eq. (2.15) with Eq. (2.16) as well as including $\nabla (\nabla \vec{v}) = \nabla^2 \vec{v}$ and $(\nabla \vec{v})^{\intercal} = \nabla (\nabla \cdot \vec{v})$, one arrives at the general Navier-Stokes momentum equation for compressible fluids

$$\frac{\partial}{\partial t} [\rho \vec{v}] + \nabla \cdot (\rho \vec{v} \vec{v}) = -\nabla p + \mu \nabla^2 \vec{v} + \frac{1}{3} \mu \nabla (\nabla \cdot \vec{v}) + \rho \vec{g} \tag{2.17}$$

2.3. Energy Equation

The conservation of energy is based on the first law of thermodynamics, which states that energy can only be transformed from one form to another, but it cannot be created or destroyed. Therefore the total energy of an isolated system remains constant. The total energy E of a material volume can be expressed as the sum of its internal and kinetic energies

$$E = m \left(\epsilon + \frac{1}{2} \vec{v} \cdot \vec{v} \right) \tag{2.18}$$

where m is the material mass and ϵ is the specific internal energy of the fluid. The total energy of a material volume changes only through the heat flow $\dot{Q}$ and performed work $\dot{W}$

$$\left(\frac{dE}{dt} \right)_{MV} = \dot{Q} - \dot{W} \tag{2.19}$$

The heat flow can be split into the flow across the surfaces $\dot{Q}_S$ and the heat generated or destroyed inside the volume, e.g. through chemical reaction, $\dot{Q}_V$. The work can be split as well: into the work performed by the surface forces $\dot{W}_S$ and work performed by the body forces $\dot{W}_b$

$$\left(\frac{dE}{dt}\right)_{MV} = \dot{Q}_S + \dot{Q}_V - \dot{W}_S - \dot{W}_b \tag{2.20}$$

The work terms are defined by

$$\begin{aligned} \dot{W}_S &= -\oint_S \left(\vec{f}_S \cdot \vec{v}\right) dS \\ \dot{W}_b &= -\int_V \left(\vec{f}_b \cdot \vec{v}\right) dV \end{aligned} \tag{2.21}$$

Replacing $\vec{f}_S$ with Eq. (2.13) as well as $\vec{f}_b$ with Eq. (2.14) leads to

$$\begin{aligned} \dot{W}_S &= -\int_V \left(-\nabla \cdot [p\vec{v}] + \nabla \cdot \left[\overline{\overline{\tau}} \cdot \vec{v}\right]\right) dV \\ \dot{W}_b &= -\int_V (\rho \vec{g} \cdot \vec{v})\, dV \end{aligned} \tag{2.22}$$

The volume and surface heat fluxes can be expressed in terms of the specific rate of heat source or sink inside the volume $\dot{q}_V$ and the specific rate of heat transfer through the surfaces $\dot{q}_S$

$$\begin{aligned} \dot{Q}_V &= -\int_V \dot{q}_V dV \\ \dot{Q}_S &= -\int_V \nabla \cdot \dot{q}_S dV \end{aligned} \tag{2.23}$$

Using these equations, one can express the conservation of energy in terms of the specific total energy $e = \frac{E}{m}$ as

$$\frac{\partial}{\partial t}(\rho e) + \nabla \cdot [\rho \vec{v} e] = -\nabla \cdot \dot{q}_S - \nabla \cdot [p\vec{v}] + \nabla \cdot \left[\overline{\overline{\tau}} \cdot \vec{v}\right] + \rho \vec{g} \cdot \vec{v} + \dot{q}_V \tag{2.24}$$

or in its integral form for a control volume V

$$\int_V \left(\frac{\partial}{\partial t}(\rho e) + \nabla \cdot [\rho \vec{v} e] + \nabla \cdot \dot{q}_S + \nabla \cdot [p\vec{v}] - \nabla \cdot \left[\overline{\overline{\tau}} \cdot \vec{v}\right] - \rho \vec{g} \cdot \vec{v} - \dot{q}_V \right) dV = 0 \tag{2.25}$$

Additionally, the energy equation can also be expressed in terms of the specific internal energy ϵ, the specific enthalpy h, and the temperature T, if the fluid is Newtonian and $h = h(p,T)$. Only the final equations are presented here, the mathematical derivation of these can be found in [26].

In terms of the specific internal energy

$$\frac{\partial}{\partial t}(\rho \epsilon) + \nabla \cdot [\rho \vec{v} \epsilon] = -\nabla \cdot \dot{q}_S - p \nabla \cdot \vec{v} + \left(\overline{\overline{\tau}} : \nabla \vec{v}\right) + \dot{q}_V \tag{2.26}$$

In terms of the specific enthalpy

$$\frac{\partial}{\partial t}(\rho h) + \nabla \cdot [\rho \vec{v} h] = -\nabla \cdot \dot{q}_S + \frac{\partial p}{\partial t} + \vec{v} \cdot \nabla p + \left(\overline{\overline{\tau}} : \nabla \vec{v}\right) + \dot{q}_V \tag{2.27}$$

In terms of the temperature

$$c_P \left(\frac{\partial}{\partial t}(\rho T) + \nabla \cdot [\rho \vec{v} T] \right) = \nabla \cdot [k \nabla T] - \left(\frac{\partial p}{\partial t} + \vec{v} \cdot \nabla p \right) \left(\frac{\partial \ln \rho}{\partial \ln T} \right)_p + \left(\overline{\overline{\tau}} : \nabla \vec{v}\right) + \dot{q}_V \tag{2.28}$$

Here, k is the thermal conductivity of the fluid, c_P the specific heat capacity at constant pressure, and : indicates a double inner product.

2.4. Dimensionless Quantities

For fluid flows, a number of dimensionless quantities have been defined in literature. They all try to characterize specific properties of the flow and are useful to compare flows with each other. Additionally they can be used to transfer measurements from a scaled down experiment to the full scale problem. The most relevant quantities to the given problem are discussed below.

2.4.1. Reynolds number

The Reynolds number measures the ratio between the advection or inertia and the diffusion or viscous forces and is defined as

$$Re = \frac{\rho v L}{\mu} = \frac{v L}{\nu} \tag{2.29}$$

where v is the velocity of the fluid relative to the considered object, L the characteristic length of the object, μ the dynamic and ν the kinematic viscosity of the fluid. The Reynolds number indicates whether a flow is laminar, transient, or turbulent. For a free flow over a flat plate, the flow becomes turbulent at $Re_x \approx 5 \times 10^5$. For a fully developed flow through a pipe, turbulent flow begins at $Re_D \approx 2300$.[39]

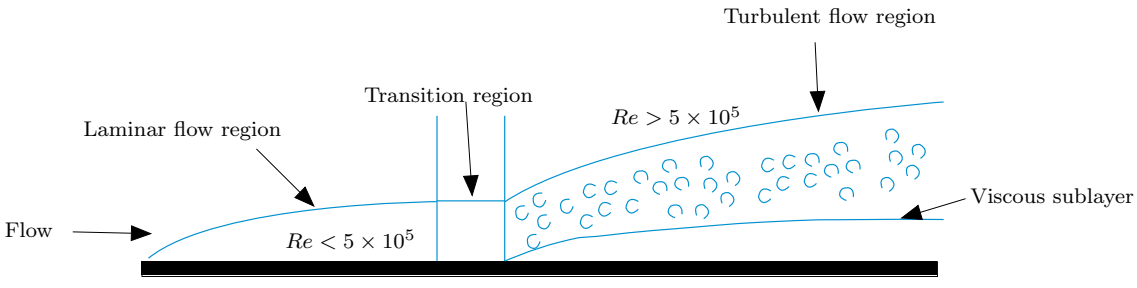


Figure 2.2.: Flow regimes over a flat plate for different Reynolds number values [26]

2.4.2. Prandtl number

The Prandtl number Pr is the ratio of the momentum diffusivity to the thermal diffusivity and can be calculated by

$$Pr = \frac{\mu c_p}{k} = \frac{\nu}{\alpha} \tag{2.30}$$

where ν is the kinematic viscosity and α is the thermal diffusivity. If $Pr < 1$, the thermal boundary layer is larger than the hydrodynamic boundary layer. If $Pr > 1$, the opposite is true.[5]

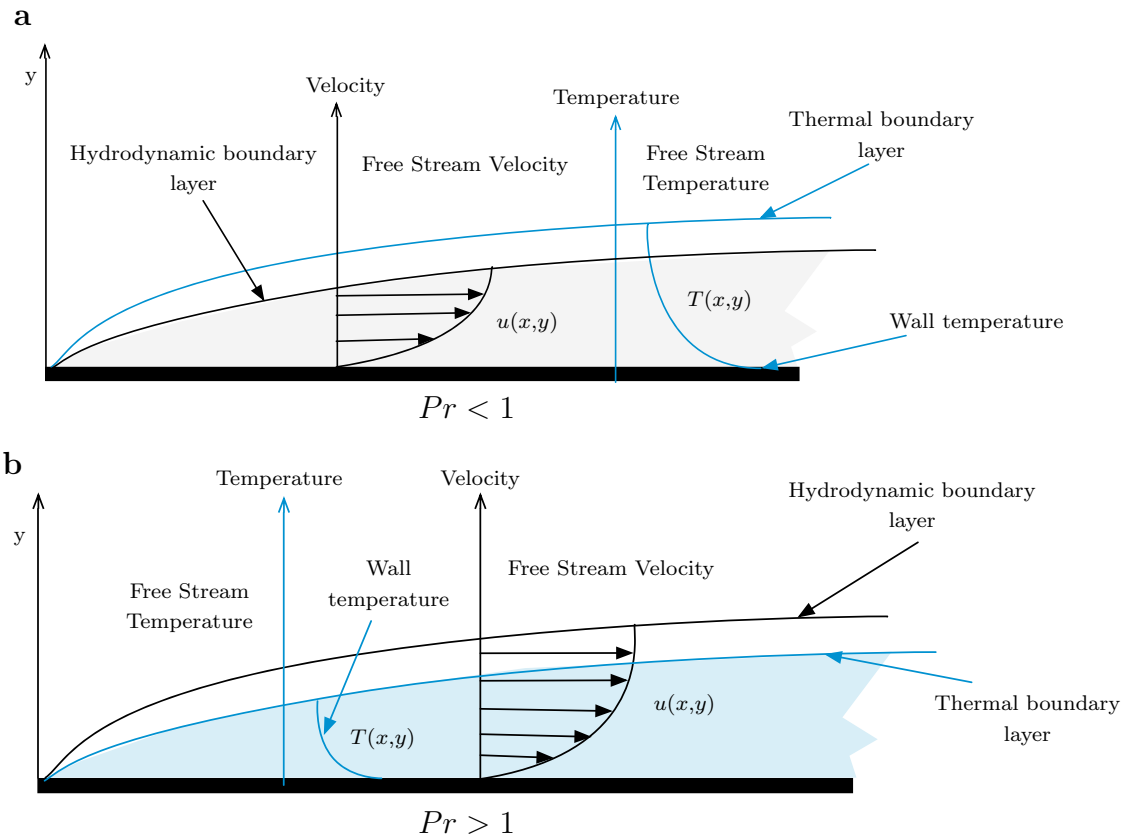


Figure 2.3.: Thermal and hydrodynamic boundary layer for **a** $Pr < 1$ and **b** $Pr > 1$ [26]

2.4.3. Grashof number

The Grashof number Gr represents the ratio between the buoyant and the viscous forces acting on a fluid. It plays a similar role in natural convection as the Reynolds number does in forced convection. It is defined as

$$Gr = \frac{g\beta(T_s - T_\infty)L^3}{\nu^2} \tag{2.31}$$

where g is the acceleration due to gravity, β is the thermal expansion coefficient, T_s and T_∞ are the temperature of the surface and the ambient environment respectively, L is the characteristic length, and ν is the kinematic viscosity of the fluid. A higher Grashof number results in a higher natural convection.[26]

2.4.4. Nusselt number

The Nusselt number Nu is the ratio of convective heat transfer to conductive heat transfer and is defined as

$$Nu = \frac{hL}{k} \tag{2.32}$$

where h is the heat transfer coefficient, L the characteristic length, and k the thermal conductivity of the fluid. A higher Nusselt number indicates a more effective convection. Typically, the Nusselt number for a laminar flow through a pipe is in the range of 1 to 10, whereas for a turbulent flow through a pipe it is between 100 and 1000.[47]

2.4.5. Rayleigh number

The Rayleigh number Ra is the product of the Grashof number and the Prandtl number

$$Ra = Gr \cdot Pr \tag{2.33}$$

It characterizes the flow regime of a fluid, with a lower value indicating a laminar flow and a higher value a turbulent flow.[5]

3. Computational Fluid Dynamics

In this chapter, the solver selection process is described and the algorithm used by the selected solver is presented. Following this, the usable numerical schemes as well as the turbulence modeling approaches are explained and compared.

3.1. OpenFOAM

OpenFOAM is the open source version of FOAM (acronym for *Field Operation And Manipulation*), which was originally developed by Henry Weller and released under the GNU General Public License in 2004.[37, 14]. The current version is distributed by the OpenFOAM Foundation while the trademark is held by the ESI-OpenCFD Ltd. The majority of maintenance and development contributions are made by the CFD Direct Ltd., a company founded by Weller in 2015.

OpenFOAM is a C++ based toolbox which allows the development of solvers for fluid and continuum mechanical problems. A variety of these solvers and libraries for different applications has been developed over the years and are shipped with OpenFOAM.

3.2. Solver selection

A solver has to meet a number of requirements to be suitable for the given application. Firstly, the solver has to be able to handle non-steady flows. Additionally, compressibility effects must be included because the fluid will be compressed during pressurization. Lastly, the solver has to be non isothermal as the fluid will heat up when being pressurization. During the solver selection process the question arose

whether the solver has to be able to model multiple fluid phases. OpenFOAM includes a number of solvers that are able to model two or more fluid phases. They use α_i as the fraction of the cell volume occupied by the fluid phase i:

$$\sum \alpha_i = \frac{\sum V_i}{V} = 1 \tag{3.1}$$

For the given problem, in addition to the requirements listed above, the solver has to be able to model phase changes due to pressure drops and increases. Unfortunately non of the multiphase solvers included with OpenFOAM fulfill all of these requirements. *interPhaseChangeFoam* looks like a promising candidate, but it can only be used for isothermal flows, which disqualifies it for the given use case.[46]

A different approach to the problem can be taken. Here, the different phases are not distinguished by the solver and instead of models for the fluid phases (e.g. ideal gas or ideal liquid), real fluid property values are used. The fluid properties are stored in tables by pressure and temperature and are interpolated by a custom OpenFOAM library during the simulation run. The fluid properties can be obtained from the NIST Chemistry WebBook.[23] In the given cases, the fluid is always either gaseous or supercritical. Therefore, there are no discontinuities in the fluid properties (see Fig. 3.1) and interpolated values can be used with a reasonable degree of accuracy.

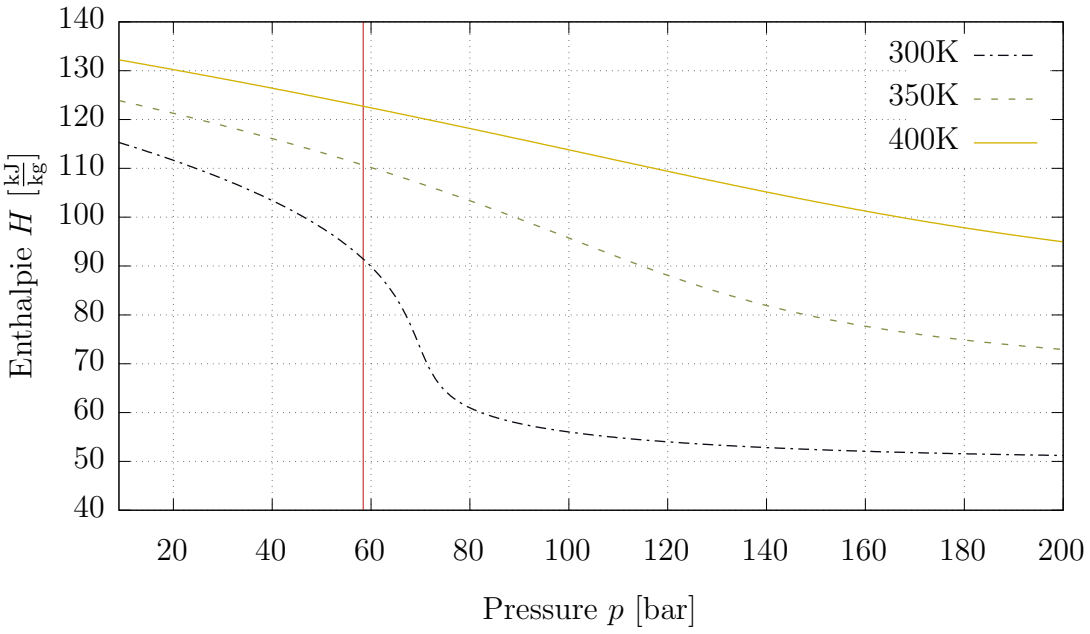


Figure 3.1.: Xenon enthalpy H over the pressure p for different temperatures T. Red line represents critical pressure [23]

By following this approach, no multiphase solver is required to solve the given problem and one of the provided compressible single phase solvers can be used. The two possible candidates are *rhoCentralFoam* and *rhoPimpleFoam*. The former is a density-based solver based on a central-upwind scheme. The latter is a pressure-based solver which uses the PIMPLE algorithm. Unfortunately, while *rhoCentralFoam* is able to solve discontinuity problems, it is not able to model turbulences in a physically correct way due to the dissipative nature of the underlying algorithm.[6] This leaves *rhoPimpleFoam* as the preferred solver.

3.3. rhoPimpleFoam

rhoPimpleFoam is solver for unsteady, compressible, non isothermal single phase fluid flows. As mentioned above, it uses the PIMPLE algorithms to solve the flow equations. This method is a combination of the SIMPLE and the PISO algorithm.[43]

3.3.1. SIMPLE

SIMPLE (acronym for *Semi-Implicit Method for Pressure-Linked Equations*) was developed by Patankar and Spalding and published in 1972.[38] SIMPLE is a pressure-based segregated method for solving steady flows. Using an initial guess for the pressure and velocity in the field, the momentum equations are solved for the relative velocity corrections. The discrete form of the momentum equation (Eq. 2.17) is then substituted into the discrete form of the continuity equation (Eq. 2.4), resulting in an equation for discrete relative pressures called pressure corrections. This is solved iteratively. Using the found correction terms, the initial guess for the pressure and velocities is updated with

$$\begin{aligned} u &= u^* + u' \\ v &= v^* + v' \\ w &= w^* + w' \\ p &= p^* + p' \end{aligned} \tag{3.2}$$

where u^*, v^*, w^* and p^* are the guesses and u', v', w' and p' are the calculated corrections.[2] Using the corrected velocities and pressure, additional transport equations

can now be solved and density field can be updated. Figure 3.2 shows a simplified flow chart of the SIMPLE algorithm.

In order for the SIMPLE algorithm to converge, under-relaxation factors have to be added to the correction equations (Eq. 3.2). These factors are typically in the range of 0.1-0.3 for the pressure and ≈ 0.7 for the velocities.[2] Additionally, the Courant number, defined as

$$C = \frac{u\Delta t}{\Delta x} \tag{3.3}$$

where u is the velocity magnitude, has to be $C \leq 1$.

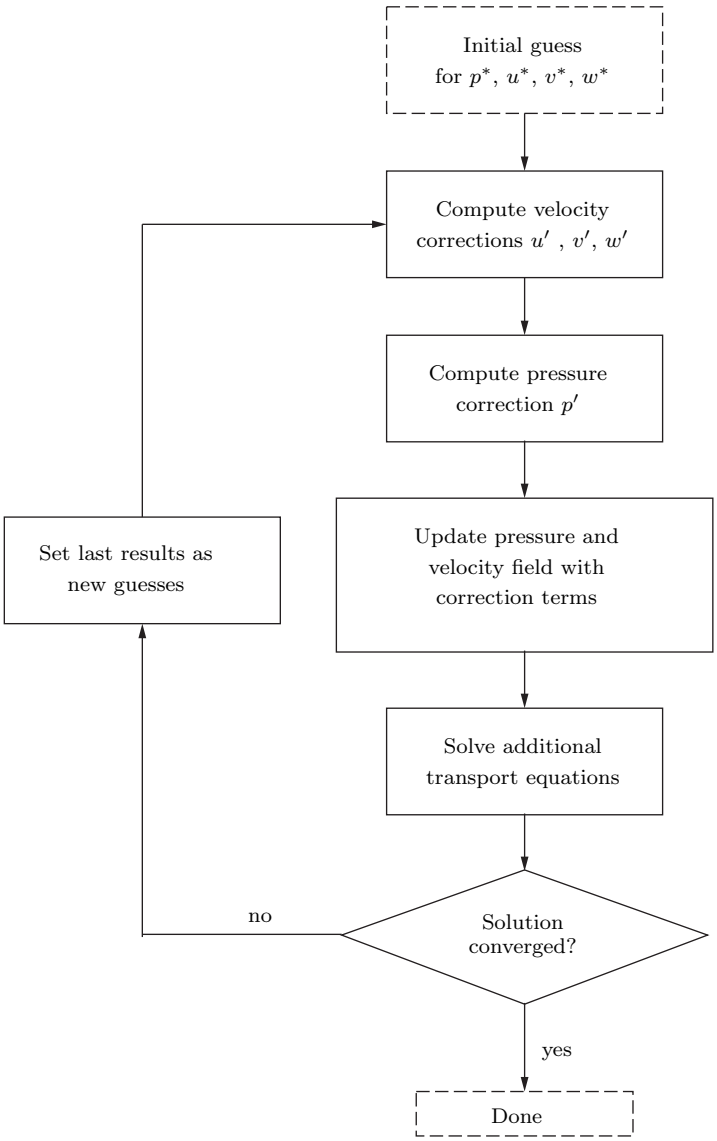


Figure 3.2.: Flow chart of the SIMPLE algorithm

3.3.2. SIMPLEC

A modified version of the SIMPLE algorithm is called SIMPLEC (acronym for *Semi-Implicit Method for Pressure Linked Equations-Consistent*). The algorithm follows the same steps as SIMPLE, but the neighbor velocity correction terms are approximated instead of dropped. The equations for calculating the velocity correction terms change to:

$$\begin{aligned}(a_e - \sum_{nb} a_{nb}) \cdot u'_e &= \Delta y(p' - p'_e) \\ (a_n - \sum_{nb} a_{nb}) \cdot v'_n &= \Delta x(p' - p'_n)\end{aligned} \tag{3.4}$$

where nb refers to the neighboring cells, e refers to the cell to the right of the current cell and n refers to the cell above the current one. The same equations can be given for the cell to the left and below the current cell. This change eliminates the need to under-relax the pressure correction term. The momentum correction still requires under-relaxation for the algorithm to be stable.[29]

3.3.3. PISO

The PISO algorithm (acronym for *Pressure-Implicit with Splitting of Operators*) was published by Issa in 1986.[19] PISO is a pressure-based method for solving unsteady flows. As with SIMPLE, an initial guess of the pressure and velocity field is required. After this, the velocity correction terms are calculated from the momentum equations and the pressure correction term is calculated using the Poisson equation. These correction terms are used to correct the pressure and velocity guesses. Then a second pressure correction term is calculated and the pressure and velocities are revised again. These values for the pressure and velocities are used to solve all other transport equations. The whole process is repeated until convergence is reached, after which the solver moves to the next time step. An overview of the process is shown in figure 3.3.

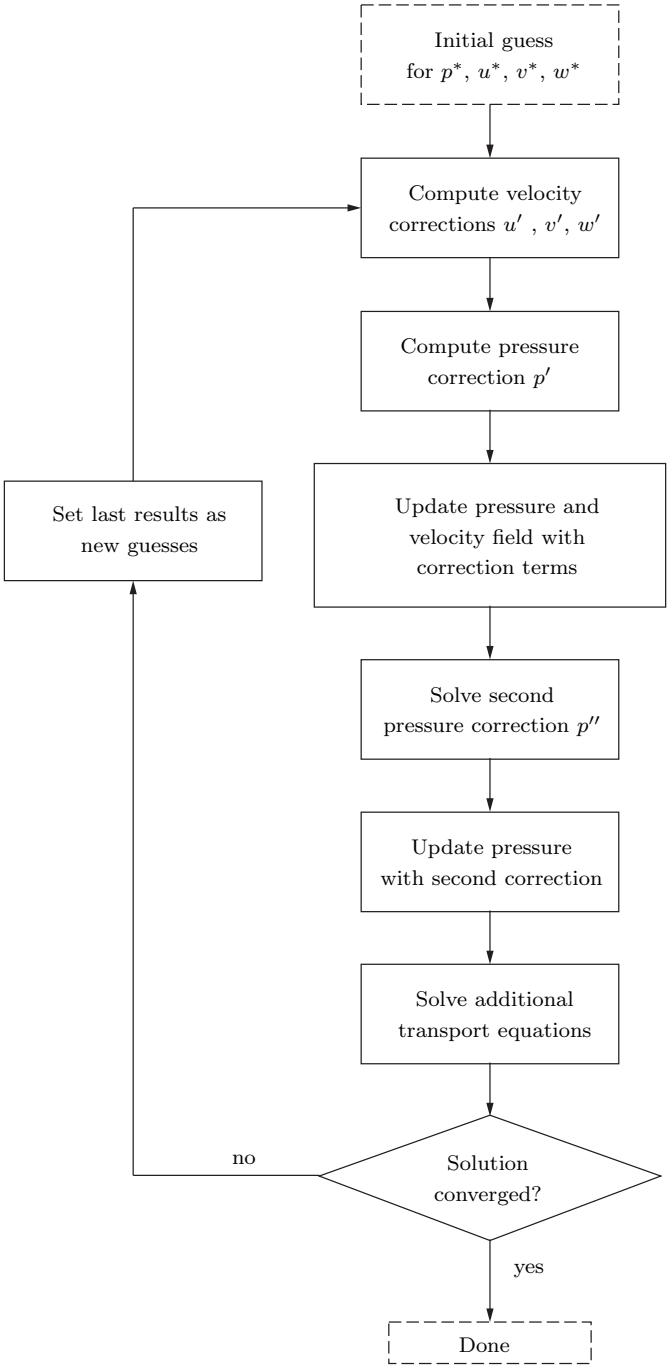


Figure 3.3.: Flow chart of the PISO algorithm for each time step

3.3.4. PIMPLE

The PIMPLE algorithm is a combination of the SIMPLE and the PISO algorithm. For each time step, a steady state solution that converges is sought after, using a specified number of correction loops. After that, all other transport equations are solved. This would be equal to the PISO algorithms with the specified number of correction loops. But following this, the algorithm loops back over the entire time step and solves the PISO algorithm again with a new initial guess.[27] A simplified flow diagram of the algorithm is shown in figure 3.4.

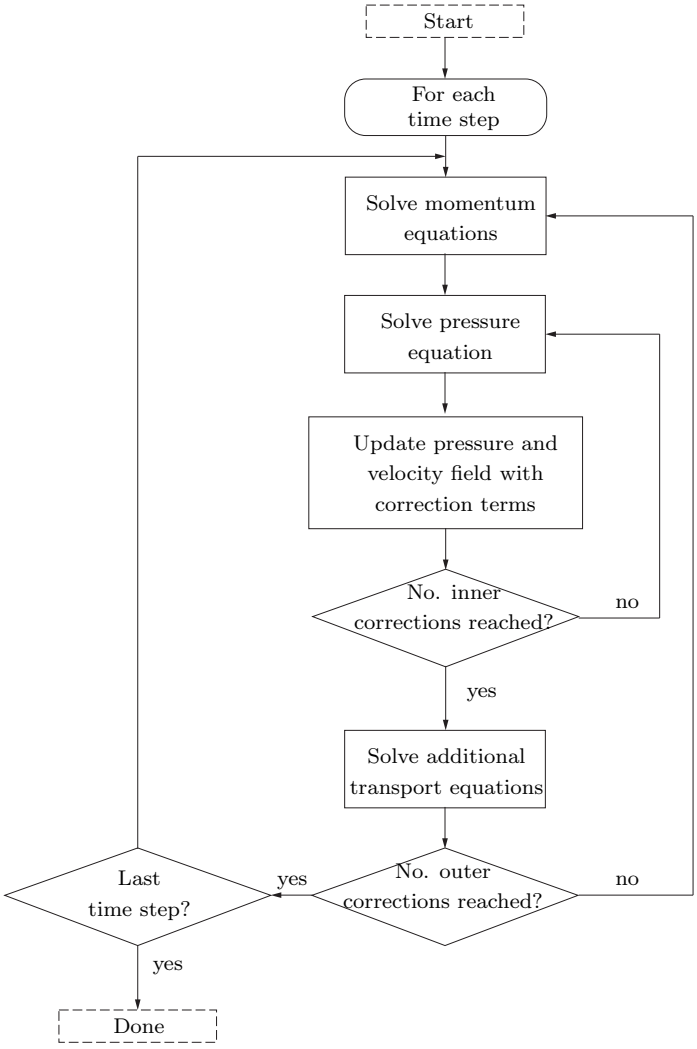


Figure 3.4.: Simplified flow chart of the PIMPLE algorithm

The usage of the PIMPLE algorithm entails the advantage that the Courant number

(Eq. 3.3) can be much larger than 1 and no under-relaxation is required for a stable solution. This allows for larger time steps, but the computational effort for each time step is bigger than with PISO.

3.4. Numerical Schemes

To solve the fluid problem, the Finite Volume method is applied. The computational domain is divided into a finite number of volumes, also called mesh. The method is based on the partial differential equation of the general law of conservation

$$\frac{\partial u}{\partial t} + \nabla \cdot f(u) = \Gamma \tag{3.5}$$

where f is the flux of the conserved state and Γ is the source term.

By applying Gauss's theorem

$$\int_V \nabla \cdot \vec{F} dV = \oint_S \vec{F} \cdot \vec{n} dS \tag{3.6}$$

to the divergence term, the change of a quantity within a volume can be described by integrating over all fluxes through the volume faces:

$$\int_{V_i} \frac{\partial u}{\partial t} dv + \oint_{S_i} f(u) \cdot \vec{n} dS = \gamma \tag{3.7}$$

where V_i is the total volume and S_i the total surface area of a cell. Different schemes can be used to discretize and solve the parts of this equation. The following sections will present the schemes available in OpenFOAM. A more detailed discussion of the schemes can be found in [26].

3.4.1. Temporal Discretization

For a transient simulation, the governing equations have to be discretized in time. To achieve this, a time coordinate is set up along which the integral of the transient term is evaluated.

In general the temporal derivative of a variable ϕ is a function of the time and the variable itself.

$$\frac{\partial \phi}{\partial t} = f(t,\phi) \tag{3.8}$$

This can be discretized in a number of ways.

3.4.1.1. Forward Euler Scheme

The Forward Euler scheme approximates the time derivative based on the current cell values. Therefore, the time derivative and the cell value at the next time step can be computed explicitly:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = f^n(t,\phi) \tag{3.9}$$

The Forward Euler scheme is a first order scheme as the truncation error scales linearly with the time step. For the Forward Euler scheme to be stable, the Courant number (Eq. 3.3) has to be ≤ 1.

3.4.1.2. Backward Euler Scheme

With the Backward Euler Scheme, the time derivative is computed from the cell values of the next time step. Thus, the method is implicit and a coupled solution algorithm is required.

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = f^{n+1}(t,\phi) \tag{3.10}$$

As with the Forward Euler scheme, the truncation error scales linearly with the time step making the Backward Euler scheme a first order scheme as well. But

unlike the Forward Euler scheme, the Backward Euler scheme is unconditionally stable.

3.4.1.3. Crank-Nicolson Scheme

The Crank-Nicolson scheme can be seen as a combination of the Forward and Backward Euler scheme. It is an implicit method.

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{1}{2}(f^{n+1}(t,\phi) + f^n(t,\phi)) \tag{3.11}$$

Because the truncation error scales quadratically with the time step, the Crank-Nicolson scheme is a second order scheme. Unfortunately, the pure Crank-Nicolson method is conditionally unstable and therefore rarely used. Instead, an off-centering coefficient Ψ is introduced. This shifts the scheme towards a Backward Euler scheme, making it more stable.

$$\Psi = \begin{cases} 1 & \text{pure Crank-Nicolson} \\ 0 & \text{Backward Euler} \end{cases} \tag{3.12}$$

A typical value for the coefficient is $\Psi = 0.9$.[46]

3.4.2. Convection Discretization

Using the Gauss theorem (Eq. 3.6), the convection terms can be computed with a surface integral over the total cell surface. Thus, the discretization schemes for the convection terms specify the method by which the face fluxes are computed.

For clarity, the schemes are presented using a one dimensional grid. But an extension to a multi dimensional grid follows the same principles. The cell, for which the fluxes shall be determined, will be referred to with the index C. The neighboring cells will be referred to by the index W for the left and E for the right neighbor. The respective faces with be referred to by lowercase letters (w, e). The cell values are represented by φ with the height of the line representing the magnitude of the value. The values of φ at the faces w and e shall be determined.

3.4.2.1. Upwind scheme

The Upwind scheme is a simple, first order scheme. The value of cell is simply assigned to the cell face depending on the fluid flow direction:

$$\varphi_w = \begin{cases} \varphi_W & \text{if} \quad u_w > 0 \\ \varphi_C & \text{if} \quad u_w < 0 \end{cases} \qquad \text{and} \quad \varphi_e = \begin{cases} \varphi_C & \text{if} \quad u_e > 0 \\ \varphi_E & \text{if} \quad u_e < 0 \end{cases} \tag{3.13}$$

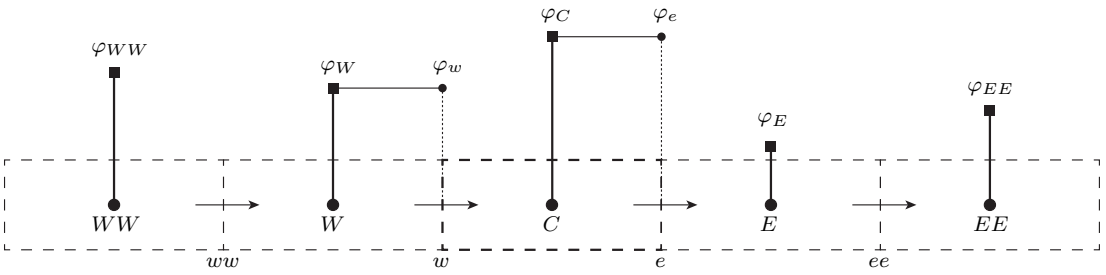


Figure 3.5.: 1D profile of the Upwind scheme

This scheme is very stable but due to to the smearing of the gradients, it only achieves low accuracy.[16]

3.4.2.2. Central Difference scheme

In the Central Differences scheme (called Linear scheme in OpenFOAM), the face value is interpolated from the two neighboring cell values:

$$\begin{aligned} \varphi_w &= \varphi_C + \frac{\varphi_W - \varphi_C}{x_W - x_C} \cdot (x_w - x_C) \\ \varphi_e &= \varphi_C + \frac{\varphi_E - \varphi_C}{x_E - x_C} \cdot (x_e - x_C) \end{aligned} \tag{3.14}$$

For a uniformed grid, this can be simplified to

$$\begin{aligned} \varphi_w &= \frac{\varphi_C + \varphi_W}{2} \\ \varphi_e &= \frac{\varphi_C + \varphi_E}{2} \end{aligned} \tag{3.15}$$

The Central Differences scheme is a second order scheme, but the directionality of the flow is not considered. Therefore, the scheme only yields physical results if diffusion is the dominant transfer mechanism over convection. Otherwise, the scheme becomes unbound and unstable.[3]

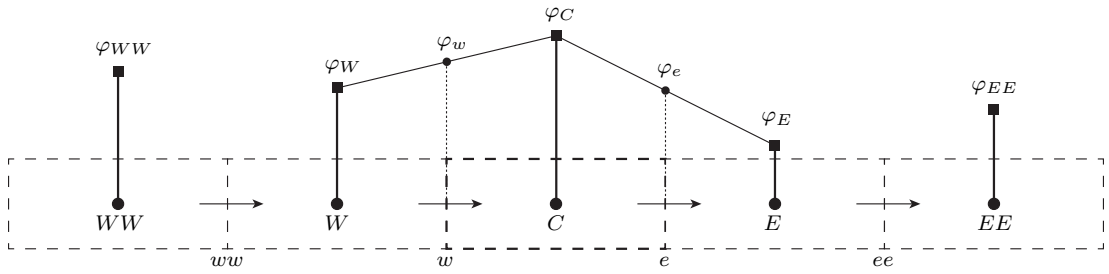


Figure 3.6.: 1D profile of the Central Difference scheme

3.4.2.3. Second Order Upwind scheme

The Second Order Upwind scheme (called Linear-Upwind scheme in OpenFOAM) can be seen as a combination of the two previous schemes. As in the case of the Upwind scheme, the cell value is applied to the upstream face. But instead of using just the closest cell value, the face value is extrapolated from the two cell values downstream:

$$\varphi_e = \begin{cases} \varphi_C + \frac{\varphi_C - \varphi_W}{x_C - x_W}(x_e - x_C) & \text{if} \quad u_e > 0 \\ \varphi_E + \frac{\varphi_E - \varphi_{EE}}{x_E - x_{EE}}(x_e - x_E) & \text{if} \quad u_e < 0 \end{cases} \tag{3.16}$$

The equivalent equation can be defined for φ_w.

For a uniformed grid, this can be simplified to

$$\varphi_e = \begin{cases} \frac{3}{2}\varphi_c - \frac{1}{2}\varphi_w & \text{if} \quad u_e > 0 \\ \frac{3}{2}\varphi_e - \frac{1}{2}\varphi_{ee} & \text{if} \quad u_e < 0 \end{cases} \tag{3.17}$$

nd the equivalent for φ_w.

As the name implies, this scheme is a second order scheme. It is less diffusive than the First Order Upwind scheme, while being more stable than the Central Differences scheme.[33]

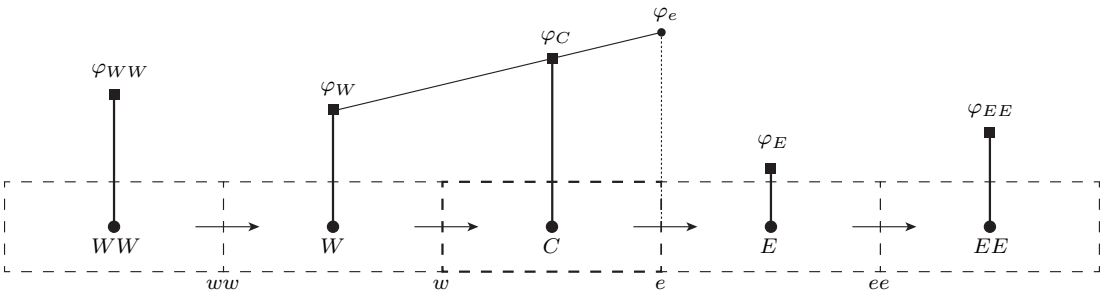


Figure 3.7.: 1D profile of the Second Order Upwind scheme

3.5. Turbulence Modeling

Turbulent flows are defined by local fluctuations of the pressure and velocity in the form of eddies on the macroscopic scale. For technical applications, their main effect is an increased diffusivity resulting in an increased heat transfer. Kolmogorov [21] describes the turbulence of a fluid as an energy cascade, with eddies of different dimensions each containing an amount of energy proportional to its size. The large eddies break up into smaller ones and transfer their energy to them in the process. This break up process continues until an eddy size is reached at which the molecular viscosity causes an effective dissipation of the energy as heat. Kolmogorov specifies this smallest size as the micro length η

$$\eta = \left(\frac{\nu^3}{\varepsilon}\right)^{1/4} \tag{3.18}$$

with ν as the molecular kinematic velocity and ε as the rate of dissipation of the kinetic energy.

The most precise method to simulate the fluid turbulences is a Direct Numerical Simulation (DNS) where the Navier-Stokes equations are solved without simplifications. This however requires a very fine mesh with spacial steps Δx smaller than the micro length η and small time steps Δt to achieve a Courant number (Eq. 3.3) $C \leq 1$. Thus, DNS is very computationally expensive and not suitable for industrial application. Instead, turbulence models are used. One of those models is the Large Eddy Simulation (LES). Here, the larger eddies are simulated directly in the same way as with DNS, but smaller eddies are approximated by a turbulence model. To achieve this, a spacial filter is applied to the Navier-Stokes equations which filtrates out everything smaller than a given filter width. The LES method

achieves high accuracies because the small eddies can be considered isotropic and can therefore be modeled very accurately.[26] To reduce the computational cost even further, the Reynolds Averaged Navier-Stokes (RANS) method can be used. Here, the flow variables are split into a mean and a fluctuating value

$$\varphi(x,t) = \bar{\varphi}(x,t) + \varphi'(x,t) \tag{3.19}$$

where the mean value $\bar{\varphi}$ calculated with

$$\bar{\varphi}(x,t) = \lim_{T\to\infty} \frac{1}{T} \int_{t}^{t+T} \varphi(x,t)\mathrm{dt} \tag{3.20}$$

To model the turbulences, a multitude of models have been developed over the years with the two-equation models $k-\varepsilon$ by Jones and Launder and $k-\omega$ by Wilcox being the most widely used ones.[26] The OpenFOAM implementation of the $k-\varepsilon$ model has been shown to be more accurate than the $k-\omega$ model [44], therefore, it was used for this work and is described in more detail below.

3.5.1. Standard $k-\varepsilon$ Model

The Standard $k-\varepsilon$ model is based on the Boussinesq approximation with the turbulence viscosity μ_t and the thermal diffusivity k_t defined by

$$\begin{aligned} \mu_t &= \rho C_\mu \frac{k^2}{\varepsilon} \\ k_t &= \frac{c_P \mu_t}{Pr_t} \end{aligned} \tag{3.21}$$

where ε is the turbulence energy dissipation rate and k is the turbulence kinetic energy.[26]

Using said model, k and ε can then be computed with

$$\frac{\partial}{\partial t}(\rho k) + \nabla \cdot (\rho \vec{u} k) = \nabla \cdot (\mu_{eff,k} \nabla k) + P_k - \rho\varepsilon \tag{3.22}$$

$$\frac{\partial}{\partial t}(\rho \varepsilon) + \nabla \cdot (\rho \vec{u} \varepsilon) = \nabla \cdot (\mu_{eff,\varepsilon} \nabla \varepsilon) + C_{\varepsilon 1} \frac{\varepsilon}{k} P_k - C_{\varepsilon 2} \rho \frac{\varepsilon^2}{k} \tag{3.23}$$

where

$$\mu_{eff,k} = \mu + \frac{\mu_t}{\sigma_k} \tag{3.24}$$

$$\mu_{eff,\varepsilon} = \mu + \frac{\mu_t}{\sigma_\varepsilon} \tag{3.25}$$

$$P_k = \rho\overline{u_i' u_j'}\frac{\partial \bar{u}_i}{\partial x_j} \tag{3.26}$$

and the following values are commonly assigned to the other model constants:

Model Constant	Value
C_μ	0.09
$C_{\varepsilon 1}$	1.44
$C_{\varepsilon 2}$	1.92
σ_k	1.00
σ_ε	1.30
Pr_t	0.90

Table 3.1.: Typical values for the $k - \varepsilon$ model constants[9]

To set up a simulation, an initial value for k and ε has to be calculated. The turbulent kinetic energy can be estimated using

$$k = \frac{3}{2}(I \cdot |u_{ref}|)^2 \tag{3.27}$$

where $I = u'/u$ is the turbulence intensity. The turbulence dissipation rate can be calculated with

$$\varepsilon = \frac{C_\mu^{\frac{3}{4}} k^{\frac{3}{2}}}{L} \tag{3.28}$$

where L is the turbulent length scale.[46]

4. Tank model

In this chapter, an OpenFOAM model in general and the created tank model in particular is described including the chosen mesh topology, thermophysical properties, and initial and boundary conditions. Additionally, two alternative modeling approaches are presented.

4.1. General Structure

The general structure of an OpenFOAM case is prescribed by the framework. [46] A case setup consists of three directories: `system`, `constant` and `0`. The `system` directory contains the files specifying the solver and the mesh setup. The `constant` folder contains information about the thermophysical and turbulence modeling properties as well as tables containing the fluid property values from NIST (see chapter 3.2). In the `0` directory, the boundary and initial condition for each flow variable at time 0 are defined.

During the simulation, an additional directory for each time step in the specified write interval is created which contains a file for each flow variable with the resulting value for each cell at this time step.

The simulation can be distributed to multiple process cores in order to speed up the calculation. In this case, each core writes its results into directory called `processorN`, where `N` is the number of this core (i.e. `processor0`, `processor1`, `processor2`, ...). After the simulation is completed, the results of all processors are combined and written into the time step directories described above using OpenFOAM's build-in tool *reconstructPar*.

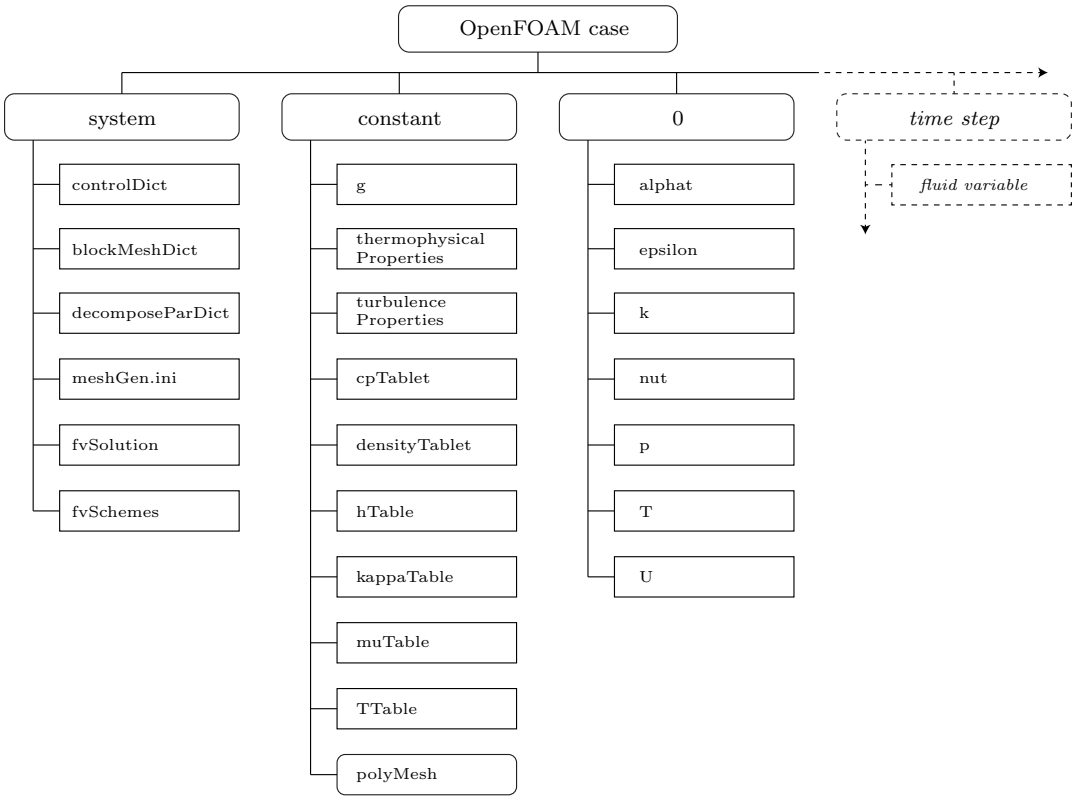


Figure 4.1.: Overview of the OpenFOAM case structure

4.2. Mesh and Solver Setup

4.2.1. Mesh

OpenFOAM always requires a three dimensional mesh, even if the model is supposed to be two dimensional. In order to create a 2D model for OpenFOAM, the third dimension is reduced to one cell.[46] Additionally, because the tank is rotationally symmetric, a wedge shape mesh can be used. This means, the two corners of the cell in the third dimension are collapsed onto each other as the symmetry axis. This results in a shape with a triangular base area (see Fig. 4.2). As previously mentioned, the dimensions of the mesh are based on Northrop Grumman's 80458-1 Xenon Propellant Tank.[34] It has a total height of 1110 mm, consisting of a cylindrical part with a height of 830 mm and a radius of 210 mm, as well as a curved dome at the top and bottom with a height of 140 mm each. The inlet/outlet

port at the bottom center of the tank has a radius of 10 mm. This is more than the tube diameter mentioned previously, but the inlet widens before reaching the inner tank walls and the diameter at this intersection point was selected. The angle of the wedge shape is defined as 0.1°. All of these physical properties are set in a file called `meshGen.ini` inside the `system` directory. This is used by a custom Python script to generate the `blockMeshDict`, which in turn is used by a standard OpenFOAM utility called *blockMesh* that creates the actual mesh and stores it inside the `polyMesh` subdirectory of `constant`.

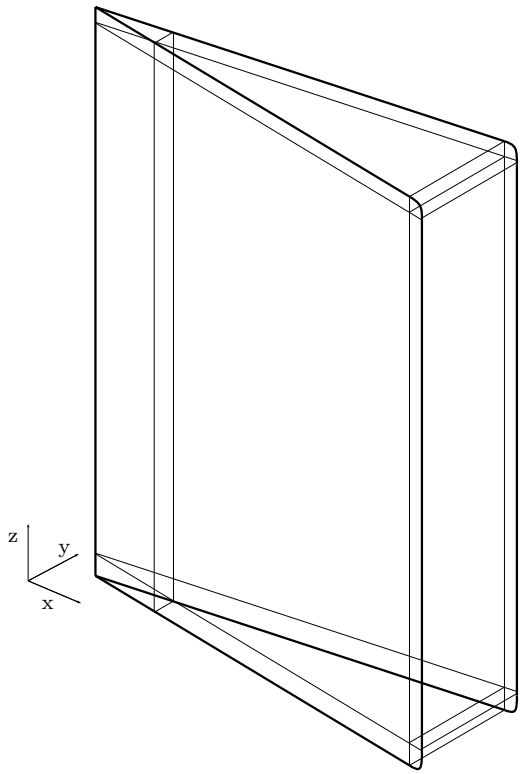


Figure 4.2.: Example of wedge shape

Additionally the `decomposeParDict` file is used for parallel calculations on multiple processor cores. It specifies the number of sections the mesh should be split into and the strategy used for the splitting process. Each section is then handed to a processor core (or thread on CPUs with multithreaded cores). Thus, the `decomposeParDict` controls the number of cores used in a parallel computation run. For this model, the `simple` stategy was used, where the domain is simply split into the specified number of equally long parts along each dimension. This is defined with

```
numberOfSubdomains 16;

method          simple;

simpleCoeffs
{
    n               (4 1 4);
    delta           0.001;
}
```

Here, `n (4 1 4)` states that the domain should be split into four pieces along the x axis, stay as one piece along the y axis and again be split into four pieces along the z axis. The `delta` value defines the acceptable cell skew factor. The amount specified under `numberOfSubdomains` has to match the number of sections created by the splitting scheme defined with `n`.

4.2.2. Schemes and Solution Algorithm Control

Inside the `system` directory, there are two files called `fvSchemes` and `fvSolution` respectively. The `fvSolution` contains the linear equation solver selections and settings as well as the PIMPLE settings (see 3.3.4). The linear equation solvers can be selected and configured for each variable independently.

For this model, the *Geometric agglomerated algebraic multigrid* (short *GAMG*) solver is used for the pressure equation with the *Simplified Diagonal-based Incomplete Cholesky* (short *DIC*) smoother. The idea behind the GAMG solver is to use a coarser grid to generate a starting solution for the finer grid. To achieve this, grid cells are combined until the specified number of cells for the coarsest level or the maximum number of combination steps is reached. The equations are then solved for the coarsest level first and the results are mapped onto the finer grids to be used as the initial solution to solve the equation for the finer grid.[4]

For all other flow variables (e.g. velocity, specific energy, etc.), the *smoothSolver* with the *Symmetric Gauss-Seidel* smoother is used. *smoothSolver* is an iterative solver which uses a run-time selected smoother for symmetric or asymmetric matrices. The *symGaussSeidel* smoother is a Gauss Seidel smoother that performs an equal number of forward and backwards smoothing sweeps, hence *symmetric*.[46]

The PIMPLE setup used consists of three inner correction loops and up to 150 outer correction loops, though the outer corrections are aborted once a residual of 5×10^{-3} for the velocity is reached. Additionally to allow for no under-relaxation,

the SIMPLEC mode (see chapter 3.3.2) is used by setting `consistent true` in the PIMPLE configuration.

The `fvSchemes` file specifies the numerical schemes to use for solving the given problem. For the time discretization, the Crank-Nicolson Scheme (see chapter 3.4.1.3) with an off-centering coefficient of 0.5 is used. For the convection discretization, the second order upwind scheme (see chapter 3.4.2.3), here called linear upwind scheme, is used.

4.3. Thermophysical and Turbulence Properties

4.3.1. Thermophysical Properties

The `thermophysicalProperties` file in the `constant` directory defines the fluid constants (e.g. molecular mass) as well as the fluid models to use for the calculations. As described in chapter 3.2, no simplified fluid model can be used for the given problem because of the different phases the fluid will be in during the simulation. Instead, the fluid properties are interpolated from tables extracted from the NIST Chemistry WebBook.[23] The `tabulatedThermophysicalProperties` extension [48] is used to pass the values from the table to the OpenFOAM solvers. The tables, called `cpTable`, `densityTable`, `hTable`, `kappaTable`, `muTable`, and `TTable` are stored inside the `constant` directory as well and tabulate the specific heat capacity c_P, the density ρ, the specific enthalpy h, the thermal conductivity κ, the dynamic viscosity μ and the temperature T respectively as a function of the pressure p and the temperature T (in the case of `TTable` as a function of p and the specific enthalpy h). The values are extracted from the isothermal and isobaric CSV files [40] provided by NIST using a custom Python script. An extract of one of these tables, in this case `densityTable`, can be found below:

```
(
(165 ((0 0) (31666.67 3.05989) (63333.33 6.18092) ...))
(168 ((0 0) (31666.67 3.00364) (63333.33 6.06383) ...))
(171 ((0 0) (31666.67 2.94859) (63333.33 5.95139) ...))
(174 ((0 0) (31666.67 2.89733) (63333.33 5.84330) ...))
⋮
)
```

4.3.2. Turbulence Properties

The turbulence model is selected and configured in the `turbulenceProperties` file. In this case, the Reynolds Averaged Navier-Stokes method (see chapter 3.5) is used and the $k-\epsilon$ model is applied to model the turbulences (see chapter 3.5.1).

4.4. Initial and Boundary Conditions

As described above, the `0` directory contains a file for each flow variable which contains the initial and boundary conditions.

4.4.1. Initial conditions

The initial flow variable values are defined using the `internalField` keyword. For the given problem, the fluid inside the tank is assumed to be at a uniform pressure and temperature, and at rest. The values for both the client and the servicer case can be found in Tab. 4.1.

parameter	unit	client	servicer
T	K	323.15	323.15
U	$\frac{\mathrm{m}}{\mathrm{s}}$	$(0\,0\,0)^\intercal$	$(0\,0\,0)^\intercal$
p	Pa	1×10^6	1.86×10^7

Table 4.1.: Initial conditions inside the tank for the client and servicer case

4.4.2. Boundary conditions

The boundary conditions are split into the port, the tank walls and the wedge sides. For the sides, the same boundary condition is used for all variables: `wedge`. This boundary condition instructs OpenFOAM to treat the sides as non-existing in a physical sense and uses the rotational symmetry to calculate fluxes across the faces.

4.4.2.1. Inlet/Outlet port

4.4.2.1.1. Pressure A simulated flow can either be pressure or velocity driven. When creating a pressure driven flow, the inlet pressure is set to a fixed value while the inlet velocity is calculated. For a velocity driven flow, it is the other way around. Through trial and error, it was found that a velocity driven flow simulation is more stable for the given problem. Therefore the pressure is not set to a fixed value (e.g. the feeding tank pressure minus a pressure drop through the pipes). Instead, a Neumann boundary condition [1] with a gradient value of 0 is used (called `zeroGradient` in OpenFOAM)

$$\frac{\partial p}{\partial n} = 0 \tag{4.1}$$

This boundary condition is physically equivalent to having a pressure regulator connected to the inlet/outlet port that regulates the pressure at the port to always be equal to the tank pressure. The regulator itself is not modeled.

4.4.2.1.2. Temperature The temperature at the inlet is not known. Therefore, a Neumann boundary condition with a gradient value of 0 is used again.

$$\frac{\partial T}{\partial n} = 0 \tag{4.2}$$

4.4.2.1.3. Velocity For a velocity driven flow, the inlet velocity has to be given. Unfortunately, the design inputs given for the problem did not specify a velocity, but a mass flow rate instead. Therefore, OpenFOAM's `flowRateInletVelocity` boundary condition is used. It calculates the flow velocity based on a given mass flow and the density at the inlet assuming that the flow is normal to the port's plane.[39] The mass flow was specified to be $3\,\frac{\mathrm{g}}{\mathrm{s}}$ for both the client and the servicer case. Because the simulated wedge only represents $\frac{1}{3600}$ of the entire tank, the mass flow rate is set to $8.\overline{3} \times 10^{-7}\,\frac{\mathrm{kg}}{\mathrm{s}}$. For the servicer case, a negative sign is used to specify an outgoing flow.

In order to simulate a valve opening at the inlet/outlet port, the mass flow is increased linearly from 0 kilogram/s at 0 s to the final $8.\overline{3} \times 10^{-7}\,\frac{\mathrm{kg}}{\mathrm{s}}$ at 1 s.

4.4.2.2. Wall

4.4.2.2.1. Pressure Just like for the inlet/outlet port, a Neumann boundary condition with a gradient value of 0 is used as a pressure boundary condition at the wall.

4.4.2.2.2. Temperature As part of the satellite, the temperature of the tank is controlled by the thermal system. The thermal control system is typically modeled using specialized software (e.g. *ESATAN* [20]) and is out of the scope of this work. Instead, a fixed wall temperature is assumed. This will then be used as an input requirement for a thermal analysis model. In our case, the wall temperature is set to

$$T_{wall} = 323.15\,\mathrm{K} \tag{4.3}$$

This is achieved using OpenFOAM's Dirichlet boundary condition [41] `fixedValue`.

The heat convection inside the fluid is modeled by the Navier-Stokes equations (Eq. 2.24). The heat convection and conduction between the fluid and the tank wall are not considered, instead the fluid at the wall is assumed to be at fixed temperature regardless of the heat transport required to achieve this.

4.4.2.2.3. Velocity The fluid cannot pass through the wall and the no-slip condition [42] must be fulfilled, meaning that a fluid boundary layer exits with a fluid velocity relative to the wall of zero at the boundary. To model this, the velocity at the wall is set using OpenFOAM's `noSlip` condition. This is a Dirichlet boundary condition with a value of

$$U_{wall} = (0\,0\,0)^{\intercal} \tag{4.4}$$

4.4.2.2.4. Turbulence One has to take care when selecting the boundary conditions for the turbulent kinetic energy k, the turbulent dissipation rate ϵ, the turbulent viscosity ν_t, and the turbulent thermal diffusivity α_t. The $k-\epsilon$ model used for turbulence modeling is only valid for fully developed turbulences, which is not the case in proximity to walls. Therefore, special boundary condition functions were developed which enable the physically correct modeling of the turbulences at a boundary wall.[24]

For the inlet/outlet port, a Dirichlet boundary condition with a value calculated using the approximation functions described in chapter 3.5.1 can be used for k and ϵ. The values for α_t and ν_t can be calculated during the run by OpenFOAM based on k and ϵ.

An overview of the OpenFOAM boundary conditions selected for the model's inlet/outlet port and wall can be found in table 4.2.

variable	in/outlet	wall
k	`fixedValue`	`kqRWallFunction`
ϵ	`fixedValue`	`epsilonWallFunction`
ν_t	`calculated`	`nutkWallFunction`
α_t	`calculated`	`compressible::alphatWallFunction`

Table 4.2.: Inlet/outlet port and wall boundary conditions used for turbulent flow properties

4.5. Comparison models

As mentioned in chapter 1, two alternative methods to model the tank will be used to compare the results. These are a 1D EcosimPro model and a 0D Excel tool.

4.5.1. Excel tool

As mentioned above, the Excel tool was created to estimate the tank shell temperature during tank pressurization on ground. A 0D approach is used and the calculations are performed solving basic equations of mass and heat transfer. The ambient temperature is assumed to be constant and the tank's heat capacity, the Prandtl number on the outside, the viscosity on the outside, and the thermal conductivity on the outside are given. The fluid pressure, specific internal energy, the specific heat capacity at constant pressure, the specific enthalpy, the viscosity, and the thermal conductivity are interpolated from 2D value tables over the temperature and the density. The values are extracted from the NIST Chemistry WebBook.[23]

The propellant mass inside the tank at time i is defined as

$$m_i = m_{i-1} + \dot{m} \cdot \Delta t \tag{4.5}$$

and the density is derived uniformly as

$$\rho_i = \frac{V}{m_i} \tag{4.6}$$

where V is the total internal volume of the tank. Using the interpolated specific internal energy h_i, the heat flow from the propellant to the tank wall can be calculated as

$$Q_{ft,i} = h_i \cdot A(T_{f,i} - T_{t,i}) \tag{4.7}$$

where A is the surface area of the tank, T_f is the temperature of the fluid and T_t is the tank shell's temperature. Considering the mass flow $\dot{m}$ into the tank and the total heat flow Q_i, the change in the internal energy of the fluid can be given as

$$\dot{U}_i = \dot{m} \cdot h_i - Q_i \tag{4.8}$$

and the internal energy can be calculated using the values from the last time step

$$U_i = U_{i-1} + \Delta t \cdot \dot{U}_{i-1} \tag{4.9}$$

Now, the fluid temperature T_f can be interpolated from the inverted specific internal energy value table. Using interpolated values for the specific heat capacity c_P, the thermal conductivity k, and the dynamic viscosity μ, the Prandtl number Pr_i, the Grashof number Gr_i, the Rayleigh number Ra_i, and the Nusselt number Nu_i can be computed (see chapter 2.4) for both the fluid-wall and the wall-environment boundary. The coefficient of thermal expansion is assumed to be $\beta = 1/T_f$.

This enables the calculation of the heat flow from the fluid to the tank wall $Q_{ft,i}$ and from the tank wall to the environment $Q_{te,i}$

$$Q_{ft,i} = h_{ft,i} \cdot A(T_{f,i} - T_{t,i}) \tag{4.10}$$
$$Q_{te,i} = h_{te,i} \cdot A(T_{t,i} - T_a) \tag{4.11}$$

Here $h_{ft,i}$ is the heat transfer coefficient between the fluid and the tank wall, $h_{te,i}$ is the heat transfer coefficient between the tank wall and the environment, and T_a is the ambient temperature.

With the total heat flow

$$Q_i = Q_{ft,i} - Q_{te,i} \tag{4.12}$$

the rate of change of the wall temperature can be written as

$$\dot{T}_{t,i} = \frac{Q_i}{C_T} \tag{4.13}$$

where C_T is the heat capacity of the tank. This finally allows for the computation of the tank wall temperature using the values from the previous time step

$$T_{t,i} = T_{t,i-1} + \Delta t \cdot \dot{T}_{t,i} \tag{4.14}$$

Applying this method, the heating of propellant as well as the tank wall during ground fueling can be estimated. Unfortunately, the results will not be directly comparable to the results produced using the 2D OpenFOAM model because the tank shell and the environment is modeled as well.

4.5.2. EcosimPro Model

As mentioned in chapter 1, EcosimPro is a 0D/1D simulation environment that can be used to model fluid flows. To simplify the model creation, libraries with prebuilt components are used, for this model in particular the *ESPSS* library.

In the beginning, a pressure and a temperature are set using a `VolPT_TMD` component. This component creates a time dependent boundary condition for the pressure and the temperature. The working fluid is specified as Xenon via a `WorkingFluid` component, using real fluid properties. The fluid is then directed through a `Jun_TMD` component, which acts as a valve and regulates the mass flow. The mass flow is set to the same $3\,\frac{\mathrm{g}}{\mathrm{s}}$ that is used in the 2D simulation. The mass flow is connected to a `Cavity` component used to model the tank. A cavity was used instead of one of the tank models included in *ESPSS* because it allows for a fixed temperature at the inner wall. The tank components all include the tank shell in their models in similar ways as the Excel tool described above. The thermal port of the cavity is connected to an external temperature component that is used to set the inner wall temperature to the same 323.15 K as in the 2D model. The sensing port is connected to a `SensorVol` component measuring the internal pressure of the tank

(cavity), which is used to update the prescribed pressure of the `VolPT_TMD` component. The second mass flow port of the cavity is connected to a dead end, effectively closing it.

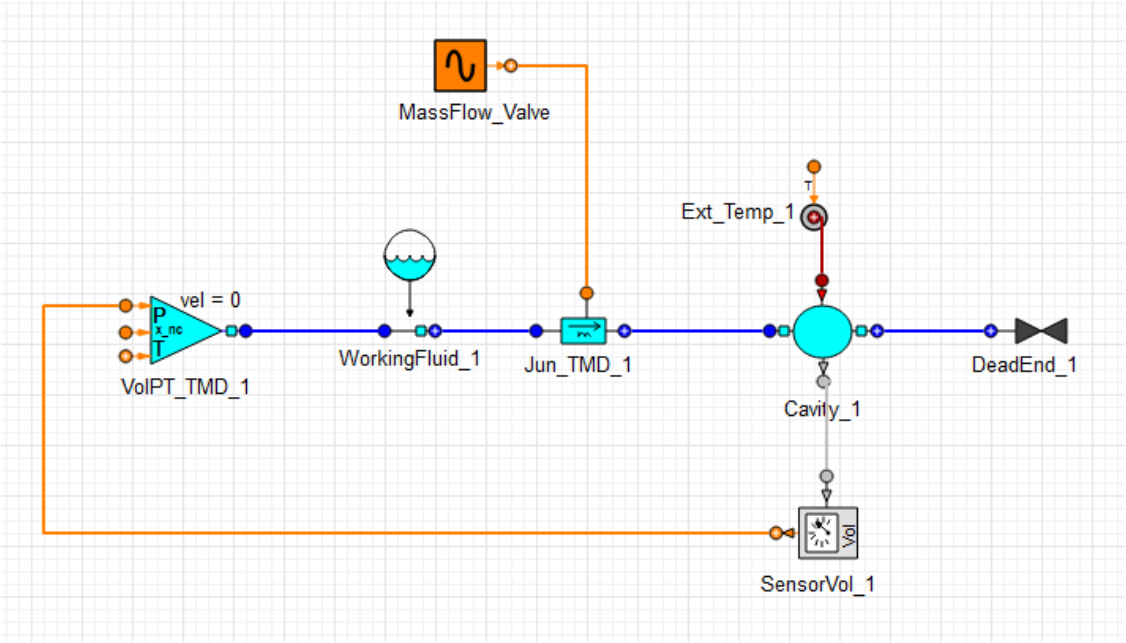


Figure 4.3.: Schematics of the EcosimPro model

5. Results

In this section, first the performed parameter study is presented and discussed. Afterwards, 2D simulation results for both the in-orbit loading (client case) and in-orbit unloading (servicer case) are presented. Finally, these results are compared to results obtained using the two previously described alternative models and the discrepancies discussed.

5.1. Parameter study

In order to find optimal values for a number of solver inputs, a parameter study has been conducted. As a baseline, the model with the boundary and initial conditions described above was used. In order to save computational effort for these studies, the calculations are performed for the first 100 seconds of the loading process and the results are then linearly extrapolated to one hour of simulated time. As a figure of merit, the time step continuity error, a measure of convergence, was used. The values are normalized with the respective median value.

5.1.1. Cell size

The cell size has a major influence on the computational effort required to calculate each time step. Assuming an equal cell size in x and z direction, the cell number quadruples when cutting the cell size in half. In this study, the cell length was varied from 20 mm to 1.25 mm in both x and z direction simultaneously.

As visible in Fig. 5.1 the calculation time increases significantly with the reduction of the cell length as expected. Meanwhile, the continuity error decreases with a smaller cell size. The normalized time $\tilde{t}$ over the mesh length can be approximated with

$$\tilde{t} = 29.025x^{-1.5396} \tag{5.1}$$

where x is the cell size in millimeters. The R-squared value of the approximation is $R^2 = 94.61\%$. The normalized error $\tilde{e}$ can be approximated with a polynomial

$$\tilde{e} = 2.782 \times 10^{-4} x^2 + 3.488 \times 10^{-3} x + 9.5 \times 10^{-1} \tag{5.2}$$

where x is the cell size again. The R-squared value of the approximation is $R^2 = 98.27\%$. Using these two functions, we find that the optimal value is somewhere around a cell length of 5 mm as the error increases almost linearly while the calculation time increases exponentially. For this reason, a cell length of 5 mm was selected for the computation.

cell size [mm]	hexahedronal cells	prismatic cells	total cells
20	542	74	616
15	927	97	1024
10	2182	149	2331
5	9026	298	9324
2.5	36700	596	37296
1.25	147992	1192	149184

Table 5.1.: Number of mesh cells for different cell sizes

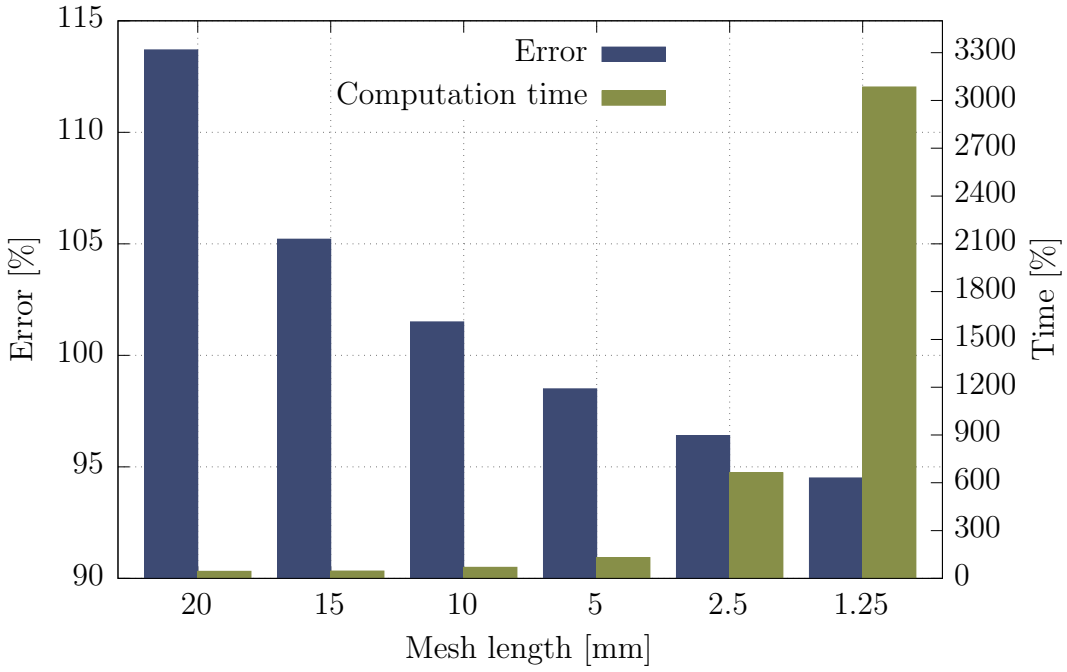


Figure 5.1.: Calculation time and continuity error for different mesh cell lengths

5.1.2. Time step

Commonly, the usable time step and cell size are liked by the Courant number (Eq. 3.3). But, since the selected solver *rhoPimpleFoam* is able to handle Courant numbers >> 1 (see chapter 3.3.4), the time step and the cell size can be varied independently of each other.

Time steps between 5×10^{-4} s and 5×10^{-2} s were evaluated. The continuity error and the calculation time for the different time steps are plotted in Fig. 5.2.

The normalized computation time $\tilde{t}$ over the time step can be approximated with

$$\tilde{t} = 4.209 \times 10^{-3} x^{-1.022} \tag{5.3}$$

where x is the length of the time step in seconds. The R-squared value of the approximation is $R^2 = 98.91\%$. The normalized error $\tilde{e}$ can be approximated roughly using a linear function

$$\tilde{e} = 9.202x + 0.9958 \tag{5.4}$$

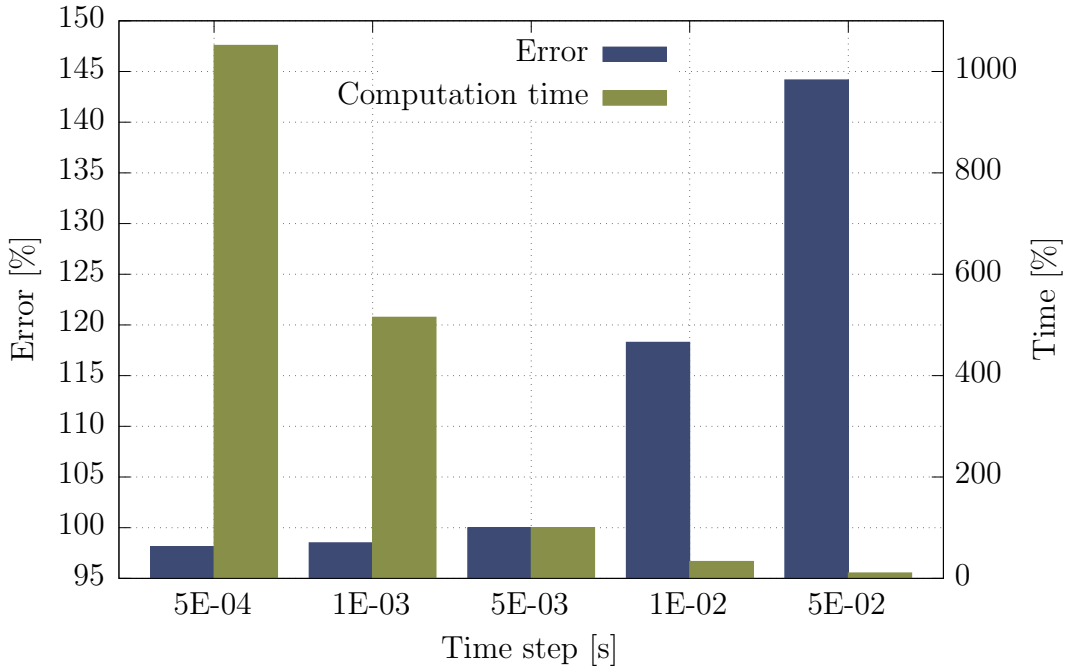


Figure 5.2.: Calculation time and continuity error for different time steps

with a R-squared value of $R^2 = 92.62\%$. x is the time step in seconds again. Using these two approximations, the optimal time step length can be found between 5×10^{-3} s and 1×10^{-2} s because the error increases roughly linearly with the time step while the calculation time increases exponentially for smaller time steps. To reduce the computation time, a time step length of 1×10^{-2} s was chosen.

5.1.3. Processor cores

The number of cores usable for the simulation and their effects on the calculation time are highly specific to the hardware used to solve the problem. In general, increasing the number of cores used for a simulation reduces the calculation time. But this relation is not linear, meaning doubling the number of cores will not halve the calculation time. This is due to the communication required between the cores after each time step. Therefore, if multiple calculations are necessary, it is advantageous to use fewer cores per simulation and run them in parallel instead of using all available cores for each simulation consecutively.

The machine used for the calculations at OHB is a server with two Intel®Xeon®Silver 4114 CPUs, each made up of 10 cores running at 2.20 GHz and having two threads per core.[18] This results in a total of 40 threads. The calculation time is evaluated

for between 1 and 24 parallel calculation threads. Both the simulation time and the total time is measured. The simulation time is the time required by OpenFOAM to complete the calculation. The total time is the sum of the simulation time and the time for pre and post processing operations that are required in order to get usable results. The data is plotted in Fig. 5.3.

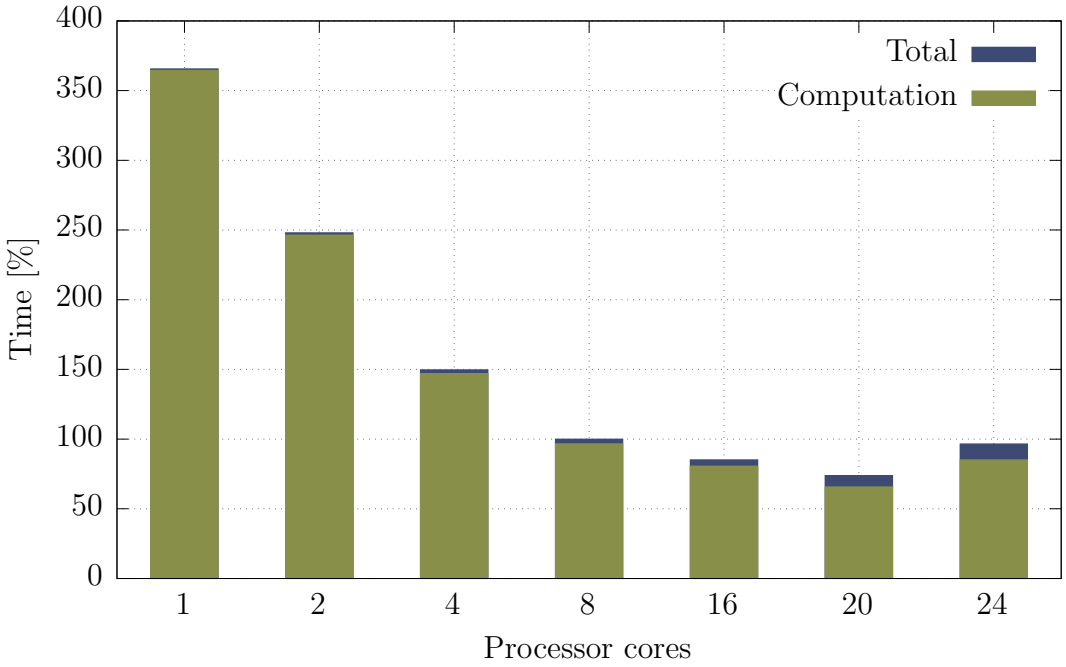


Figure 5.3.: Calculation time for different number of computing threads

For the range between 1 and 20 threads, the normalized total calculation time $\tilde{t}$ can be approximated by

$$\tilde{t} = 3.483x^{-0.544} \tag{5.5}$$

where x is the number of threads. The R-squared value of this approximation is $R^2 = 98.37\%$. As visible in Fig. 5.3, the time increases again for a thread number larger than 20. This is due to the fact that the processors in the used machine have only 20 physical cores, as described above. Traditionally, a processor core had one thread of execution. However, this meant that the core was idle when it was waiting for a high-latency operation, such as reading from memory or worse from the hard drive. To reduce the idle time, a second execution thread was introduced and the processor core switches to it when the first one is waiting for a high-latency event. Unfortunately when running an OpenFOAM calculation, the parallel processing threads have to wait for each other after every time step and exchange information

about the boundaries between their respective domains. When using more parallel threads than the number of processor cores present in the machine, two threads have to be handled by the same core. Since one of these threads will finish before the other, all remaining threads have to wait for the last one to finish. This additional wait results in an increased calculation time. Therefore, a parallel OpenFOAM calculation should never use more threads than the number of cores in the used machine, in our case 20.

Additionally, an increase in the post processing time can be observed with a rising number of computing threads. This is a result of the additional file reads required to recombine the results of each section into the results directories for the entire domain. The actual reconstruction time and time increase with additional threads depends heavily on the speed of the used storage hardware and the utilization by other users at the same time.

For our calculations, between 4 and 16 threads were used, depending on the number of calculations performed at the same time and the utilization of the server by other users. 20 threads were not used as the increased post processing time results in very small time savings compared to 16 threads.

5.2. Client case

The first case that was examined is the client case for the *ESPRIT* project. In this case, the xenon tank will be pressurized in orbit. As previously mentioned, the tank is assumed to have an initial internal pressure of $p_0 = 1 \times 10^6\,\mathrm{Pa}$ and a temperature of $T_0 = 323.15\,\mathrm{K}$. Because of the time intensive nature of the simulation, only the first hour of the propellant loading process is simulated.

Fig. 5.4 shows a visualization of the pressure at multiple time steps. The pressure inside the tank increases uniformly. As we have a subsonic flow and therefore no shock waves, this is the expected behavior. A plot of the pressure inside the tank over time can be seen in Fig. 5.5.

The temperature of the xenon inside the tank over time can be looked at as a visualization at multiple time steps (Fig. 5.7) and as a graph of the average temperature over time (Fig. 5.6) below. The average fluid temperature rises as first because the energy transferred into the tank due to the pressurization is more than the energy extracted through the walls. As the process continues, the fluid temperature decreases until an equilibrium is reached. The wave pattern above the inlet observable in the temperature visualizations, especially between 400 and 1500 s is a numerical artifact from the off-centering coefficient value chosen for the

Crank-Nicolson Scheme (see chapter 3.4.1.3). A value of $\Psi = 0.5$ was selected, which biases the scheme towards a Backward Euler Scheme. Using higher value for Ψ, thereby shifting the scheme towards a pure Crank-Nicolson scheme, removes the numerical artifacts but results in the simulation becoming unstable. Because the waves are locally bound to the inflow close to the symmetry line, they have a negligible effect on the average fluid temperature and the heat flow across the wall boundary. Thus, the selected off-centering value of $\Psi = 0.5$ is acceptable and should be used to increase stability.

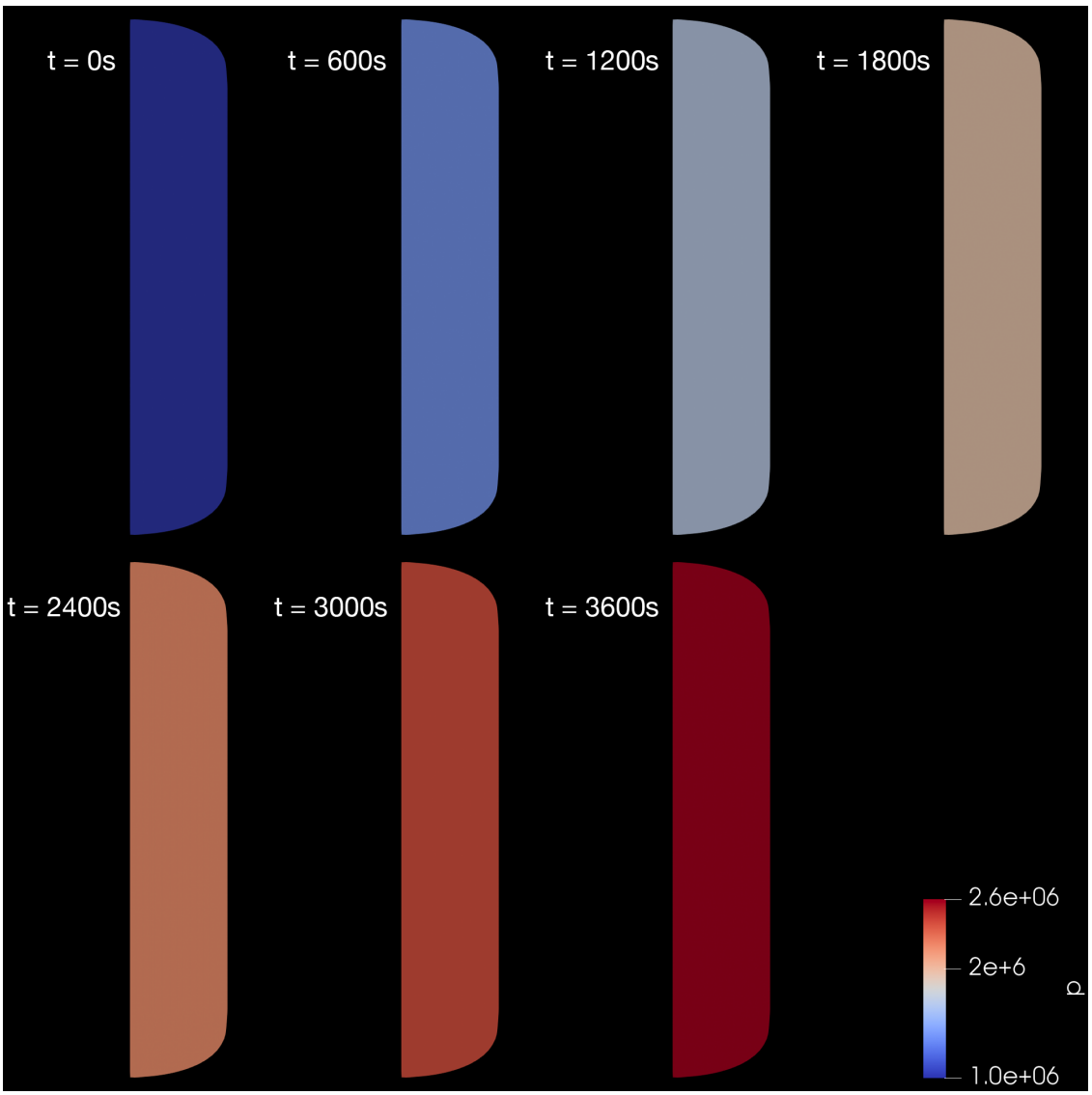


Figure 5.4.: Fluid pressure [Pa] inside the tank at different times

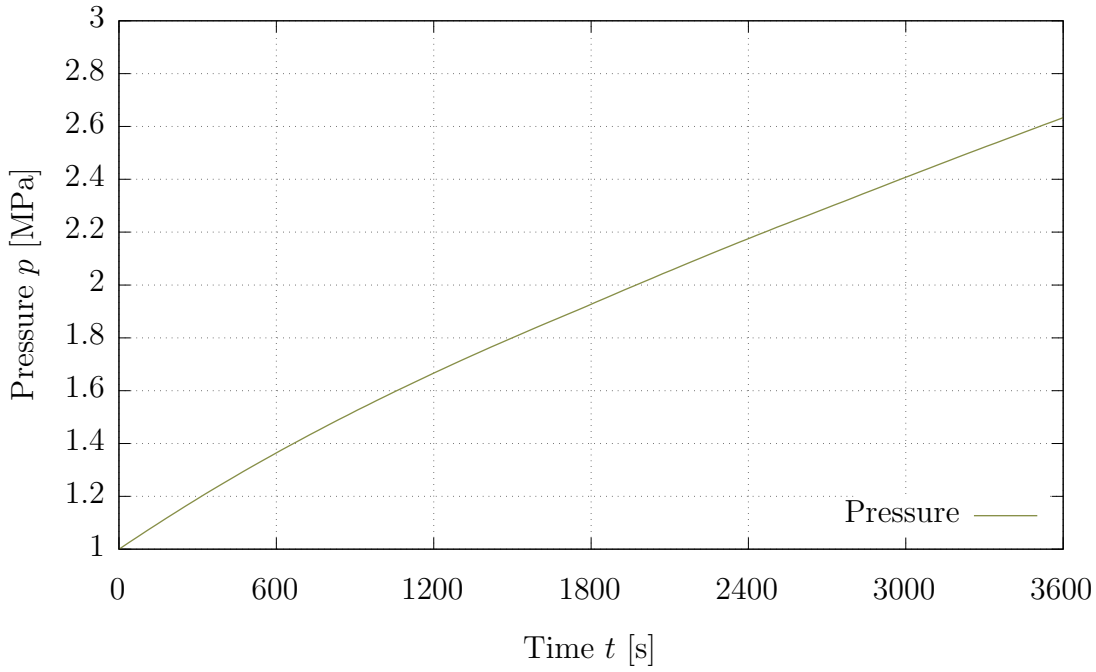


Figure 5.5.: Pressure of the fluid inside the tank over time

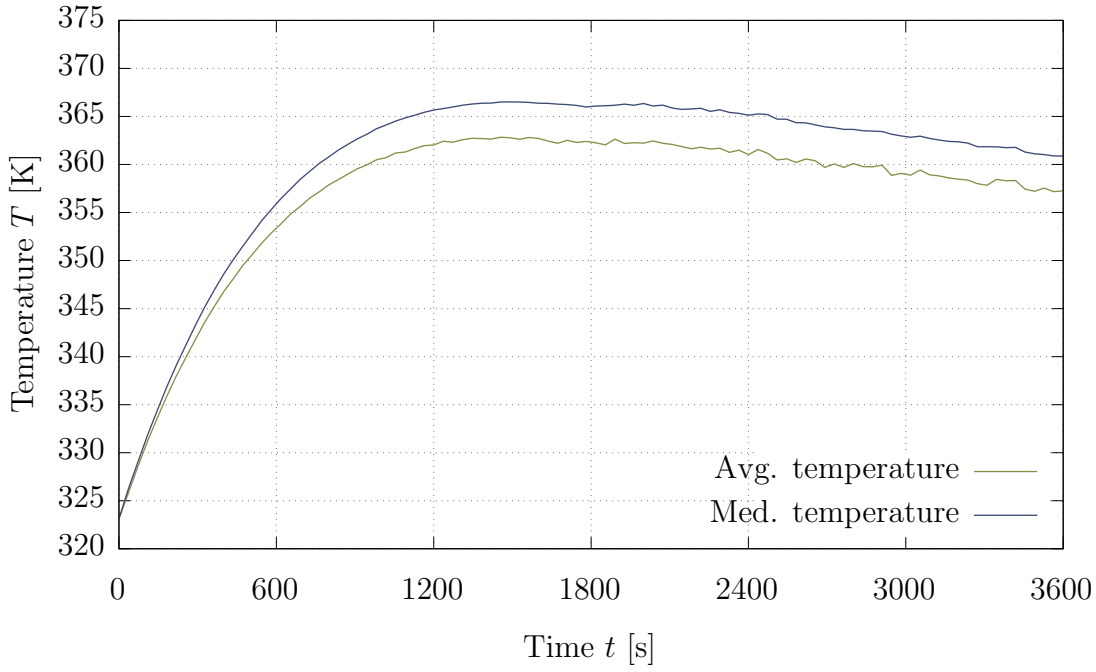


Figure 5.6.: Average temperature of the fluid inside the tank over time

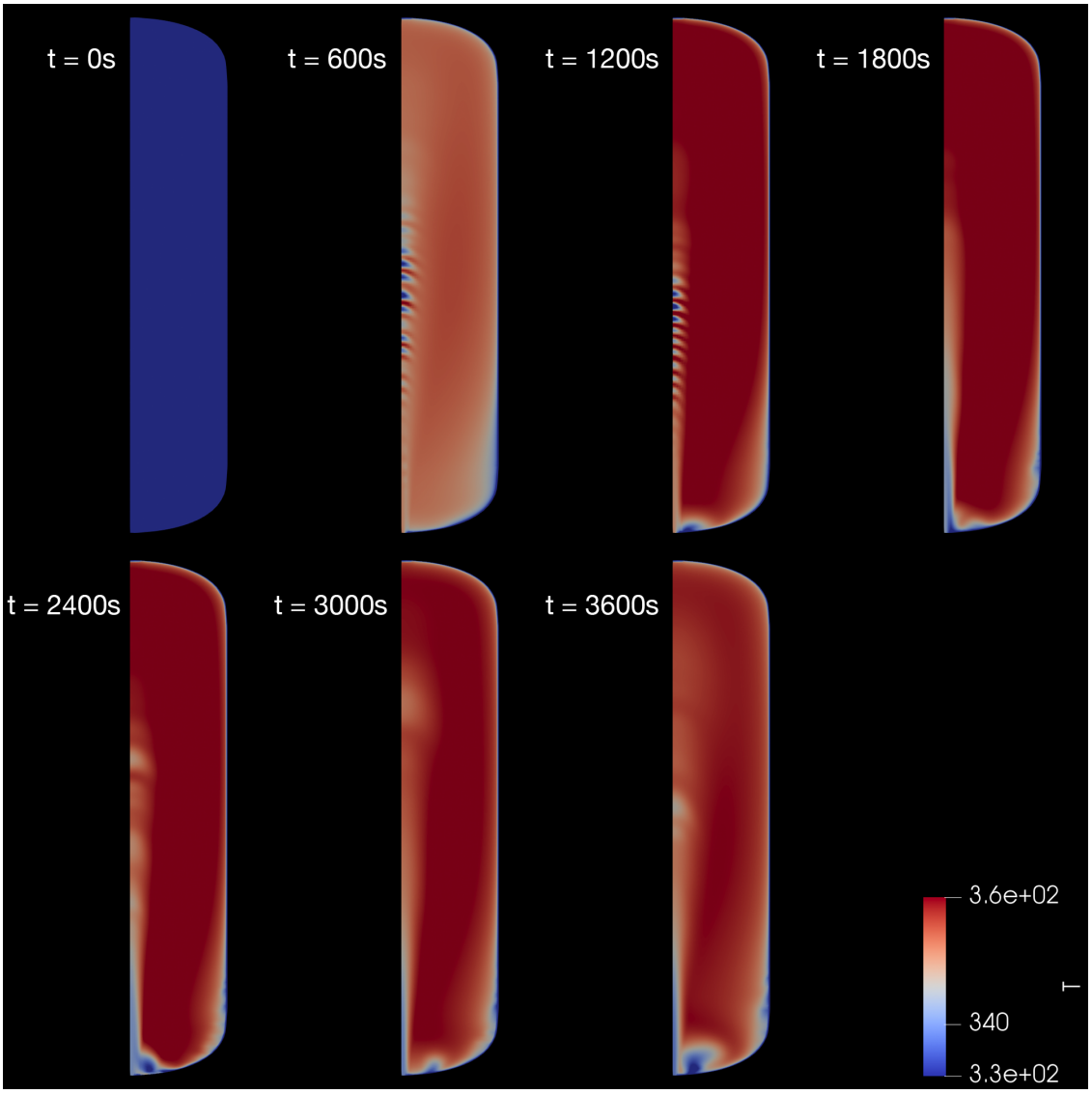


Figure 5.7.: Fluid temperature [K] inside the tank at different times

A visualization of the velocity magnitude can be seen in Fig. 5.8 and a plot of the average velocity over time is shown in Fig. 5.9. Overall, the fluid inside the majority of the tank is almost stationary. The velocity at the inlet is relatively small as well with a maximum value of 0.187 $\frac{\mathrm{m}}{\mathrm{s}}$ and is decreasing over time as the density of the inflowing propellant increases.

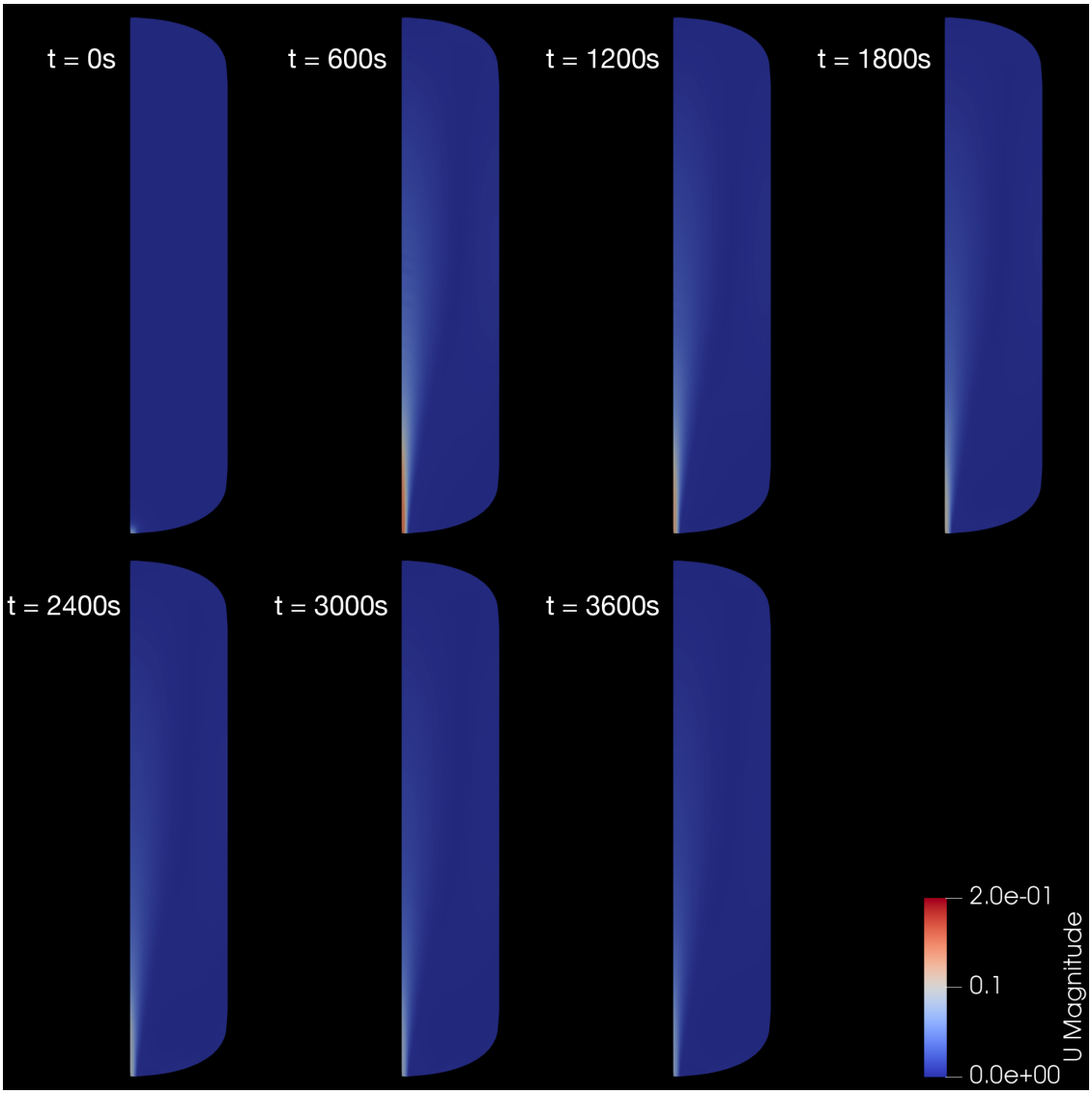


Figure 5.8.: Fluid velocity $[\frac{\mathrm{m}}{\mathrm{s}}]$ inside the tank at different times

Using OpenFOAM's *wallHeatFlux* utility, the heat flow over the tank wall required to keep the fluid temperature at the wall to the given fixed value of 323.15 K can be calculated. The heat flow over time is shown in Fig. 5.10. The heat flow rises as the fluid temperature inside the tank increases until an equilibrium is found.

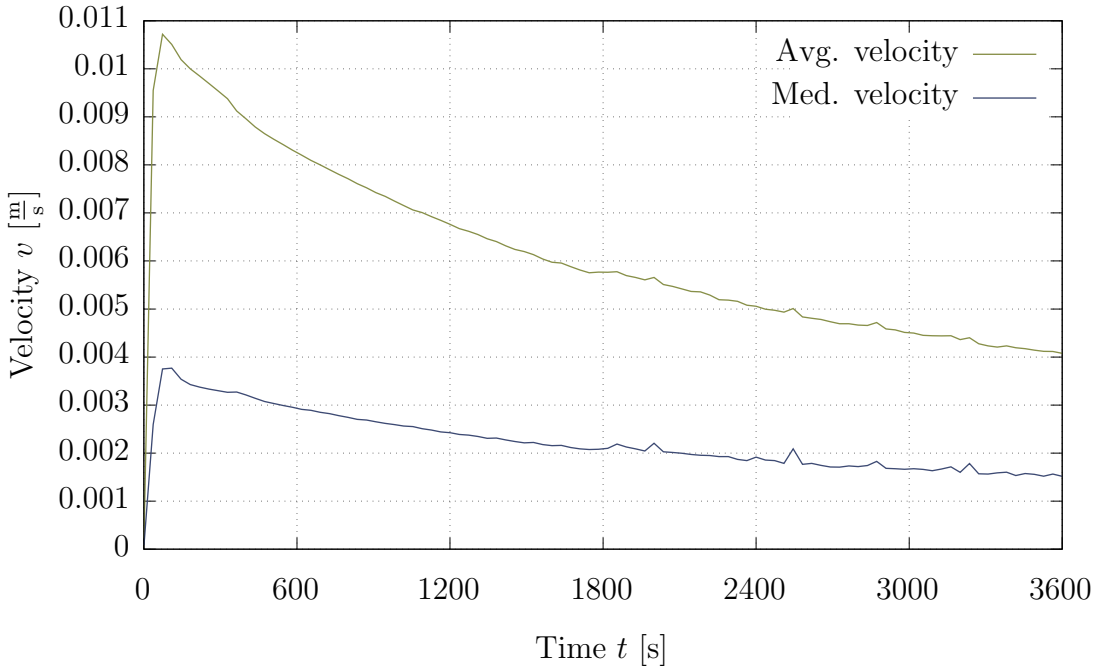


Figure 5.9.: Average velocity magnitude of the fluid inside the tank over time

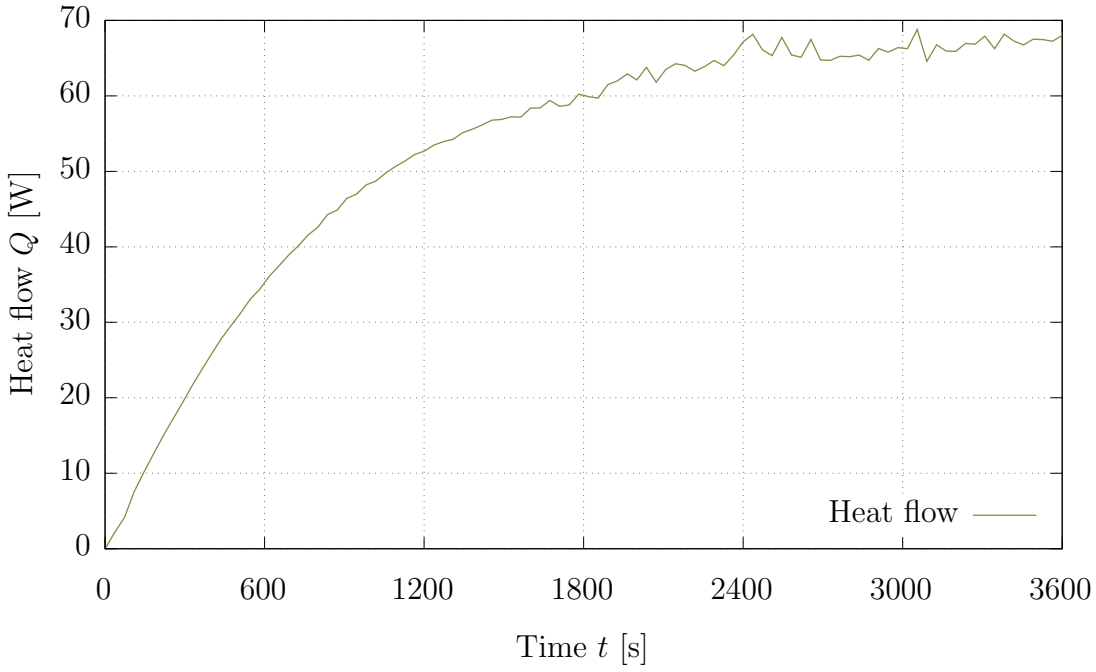


Figure 5.10.: Heat flow out of fluid through the tank wall over time

5.2.1. EcosimPro comparison

The results of the OpenFOAM simulation are compared with the results produced by the EcosimPro model described in Chapter 4.5.2. Fig. 5.11 shows the EcosimPro and the OpenFOAM simulations agreeing closely with each other for the pressure inside the tank. The average fluid temperature inside the tank, seen in Fig. 5.12, is less consistent between the OpenFOAM and the EcosimPro simulation. Here, EcosimPro underestimates the temperature increase compared to OpenFOAM. A possible reasons for this discrepancy is that the EcosimPro assumes all fluid parameters to be uniform inside the tank. This assumption is physically correct for the pressure, but as visible in Fig. 5.7, it is incorrect for the temperature. This can result in a more efficient heat transfer through the tank wall, reducing the temperature increase of the fluid. It is also possible that the OpenFOAM boundary conditions are unable to be transferred to a 1D/0D simulation, as they require the fluid temperature at the wall to be fixed without constraining the overall fluid temperature directly.

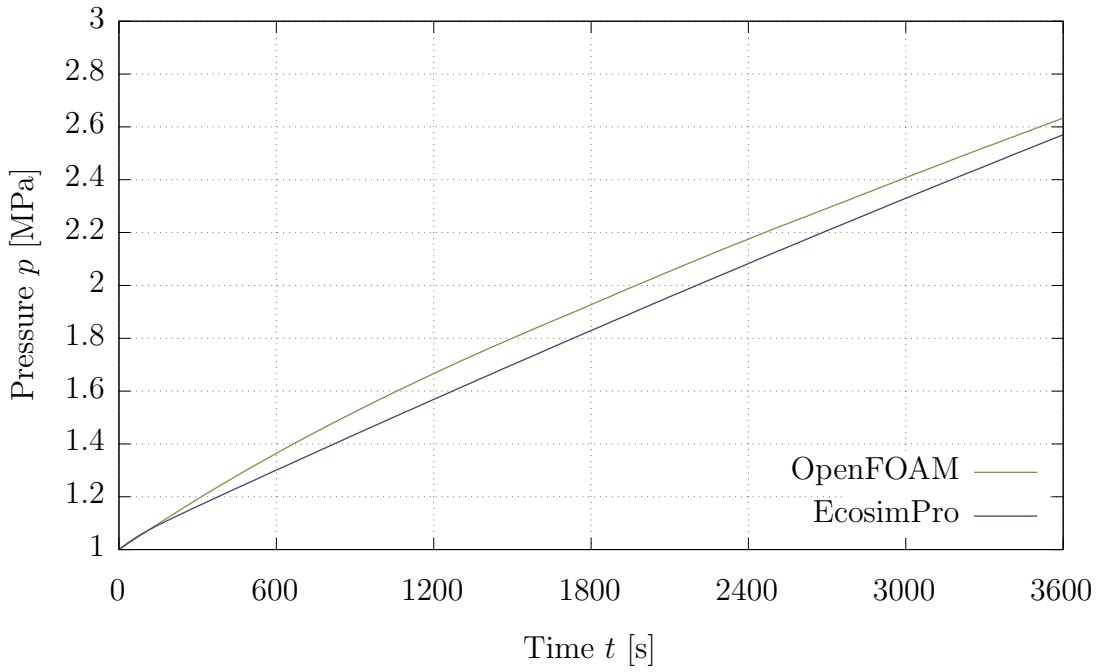


Figure 5.11.: Fluid pressure according to the OpenFOAM and the EcosimPro simulation over time

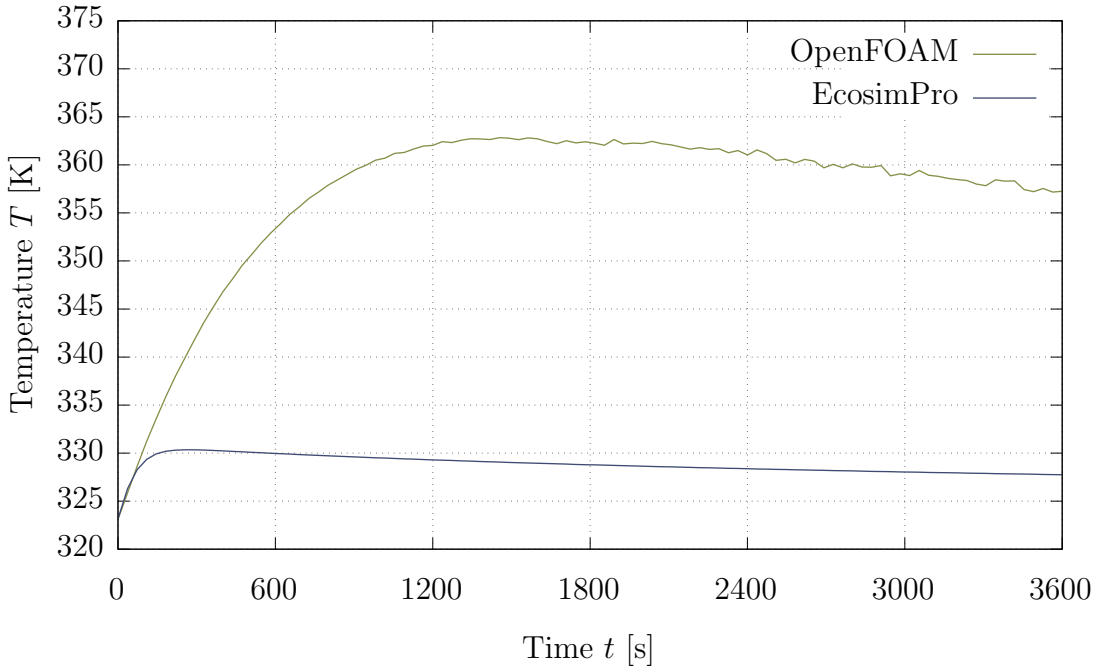


Figure 5.12.: Average fluid temperature according to the OpenFOAM and the EcosimPro simulation over time

5.2.2. Excel tool comparison

The 0D model described in chapter 4.5.1 models the tank wall in addition to the fluid itself. Additionally the temperature of the wall is not kept at a fixed value and the tank shell is cooled convectively with a fixed ambient temperature.

For this reason, a direct comparison between the results of the OpenFOAM simulation and the Excel calculations is not possible. Instead, the Excel tool can be used to compare the order of magnitude of the heat flow. By varying the ambient temperature, a quasi-constant value for the wall temperature in the first hour of the pressurization process can be found. When using this method to set the wall temperature to $\approx 323.15\,\mathrm{K}$, an average heat flow of $46.17\,\mathrm{W}$ within the first hour is predicted. The corresponding calculations by OpenFOAM yield a value of $52.81\,\mathrm{W}$. The small discrepancy between the two values can be explained by the difference in the boundary conditions. In the Excel tool, the tank wall temperature was controlled to roughly $323.15\,\mathrm{K}$ whereas the OpenFOAM simulation uses a fixed fluid temperature at the wall.

5.3. Servicer case

The second case of interest is the servicer case of the *ESPRIT* project. Here, the tank will be used as a propellant source in orbit, effectively depressurizing it. As previously mentioned, the tank is assumed to have an initial internal pressure of $p_0 = 18.6 \times 10^6\,\text{Pa}$ and a temperature of $T_0 = 323.15\,\text{K}$. Once again, the first hour of the propellant unloading process is simulated.

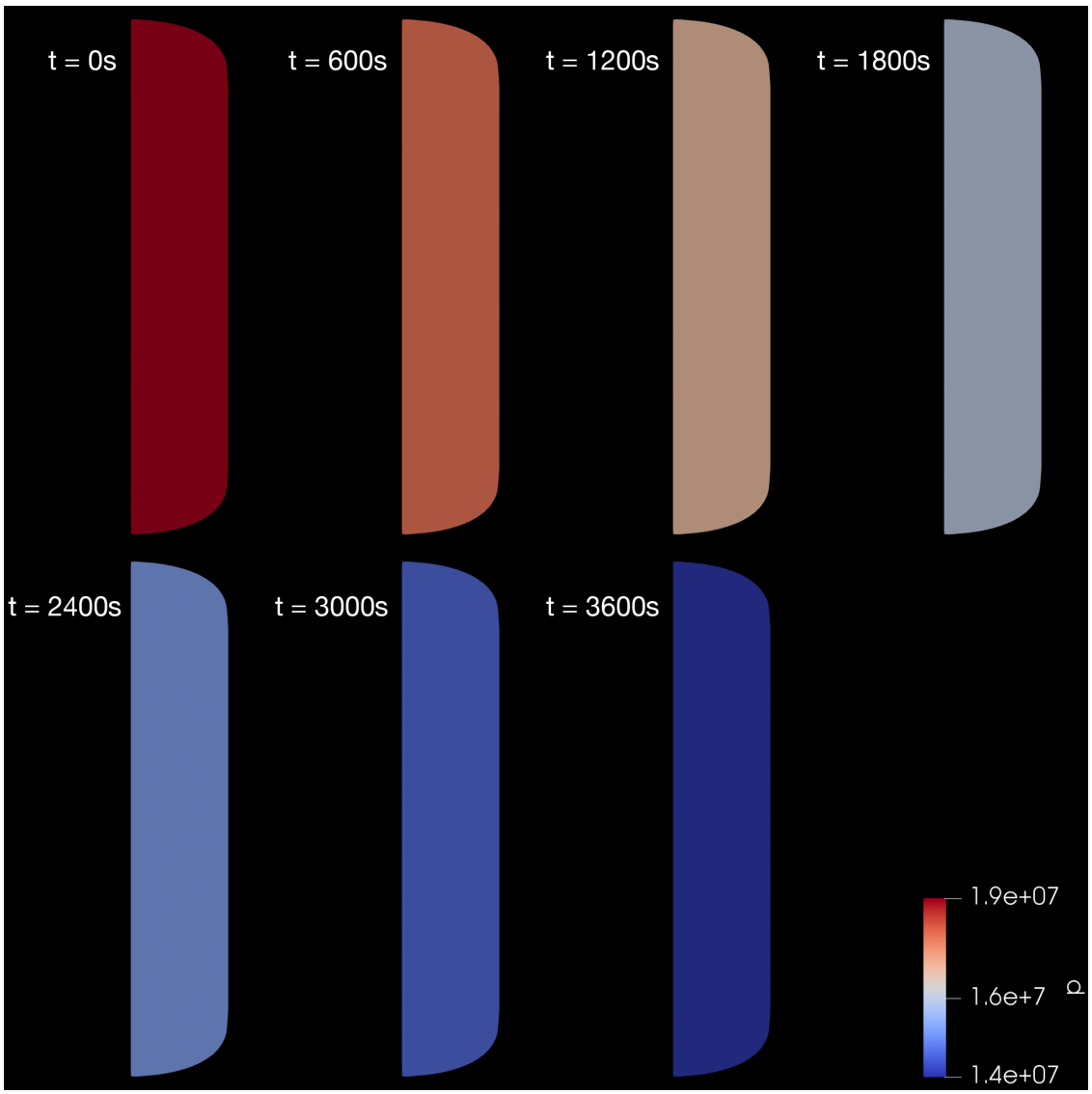


Figure 5.13.: Fluid pressure [Pa] inside the tank at different times

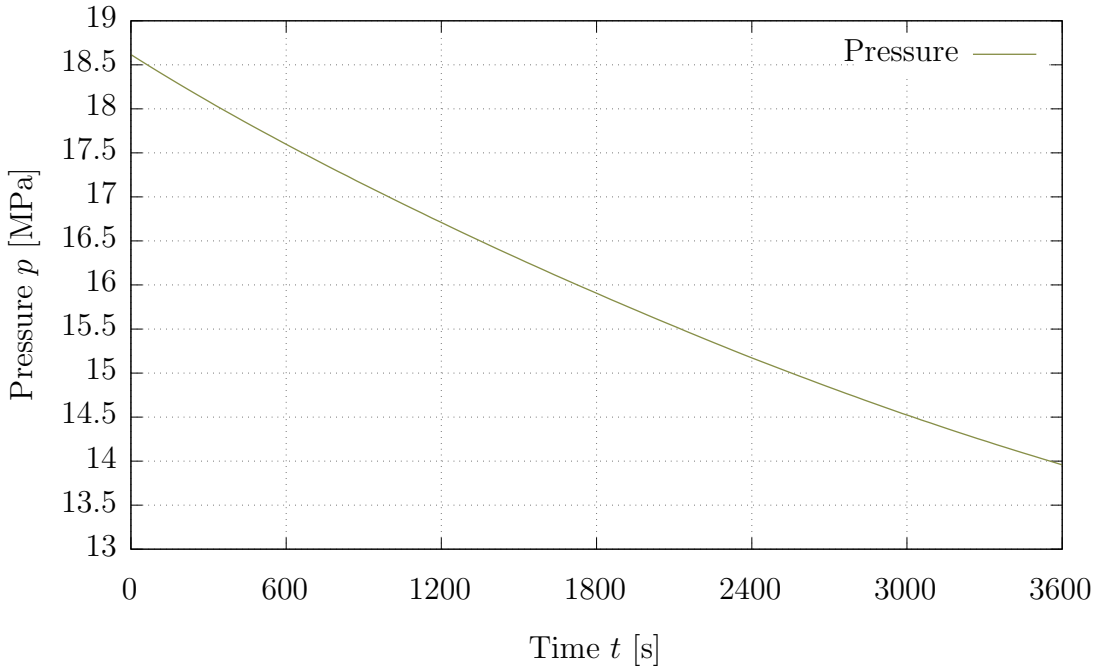
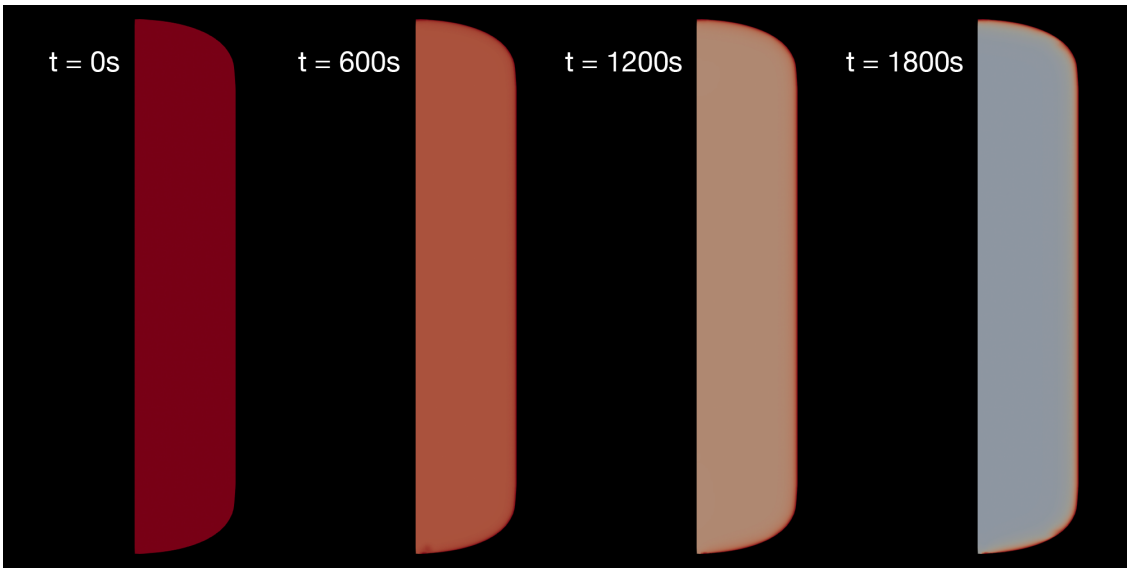


Figure 5.14.: Pressure of the fluid inside the tank over time

The pressure decrease inside the tank is visualized in Fig. 5.13 and plotted over time in Fig. 5.14. Similar to the client case, the pressure distribution inside the tank is uniform due to the subsonic nature of the flow.



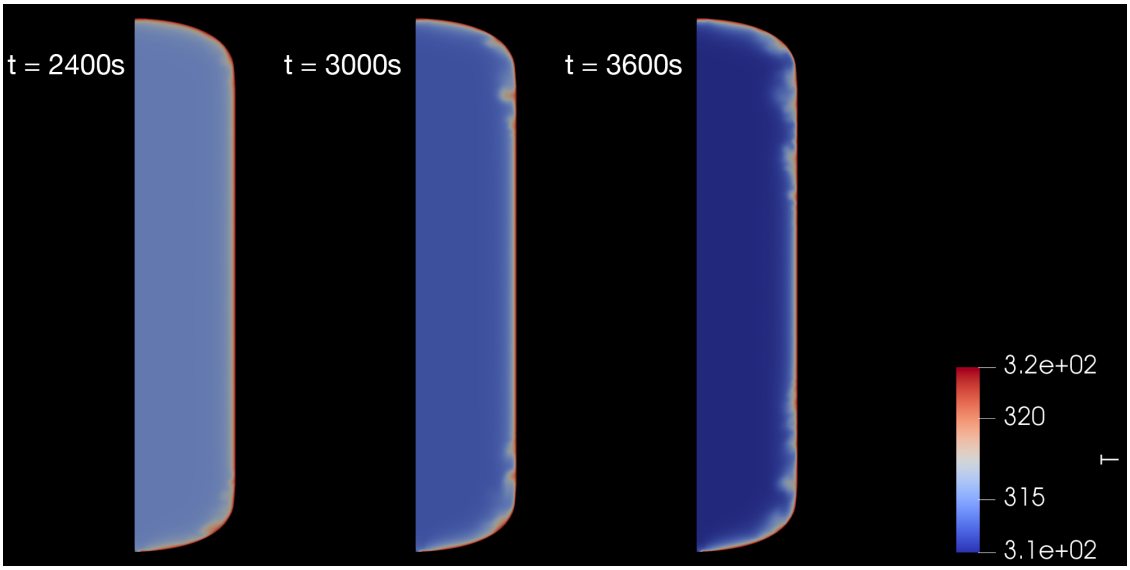


Figure 5.15.: Fluid temperature [K] inside the tank at different times

The development of the fluid temperature inside the tank is shown in Fig. 5.15. As the temperature decreases due to the gas expansion, the heating effect from the fixed wall temperature becomes visible. With an increasing temperature difference, local temperature variations at the wall boundary can be observed. These variations are diffused over time due to the natural conduction and convection processes of the fluid. The average and median temperature of the propellant is plotted over time in Fig. 5.16.

A visualization of the fluid velocity magnitude is shown in Fig. 5.17. Overall, the fluid inside the tank is very close to stationary, with the average velocity, plotted in Fig. 5.18, between 1.5 and $2.5 \times 10^{-5} \frac{\mathrm{m}}{\mathrm{s}}$. After an initial spike due to the valve opening, the temperature rises as the pressure deceases due to the reduced density of the outflowing propellant.

Using the OpenFOAM utility *wallHeatFlux* again, the heat flow into the propellant through the tank wall required to hold the boundary temperature can be calculated. The heat flow over time is shown in Fig. 5.19.

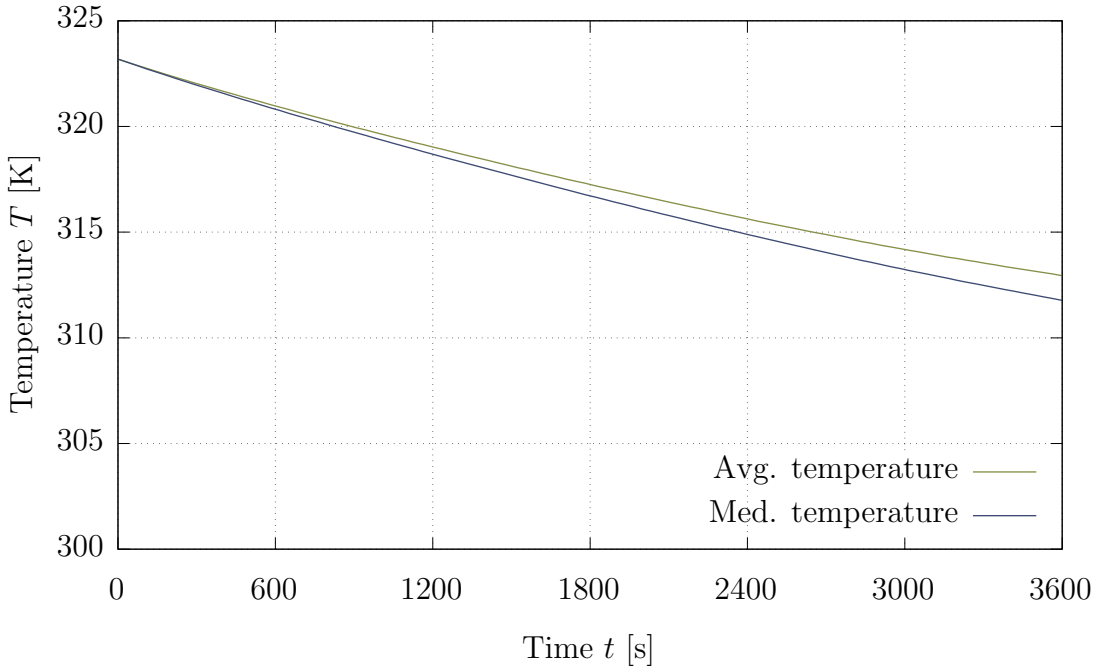
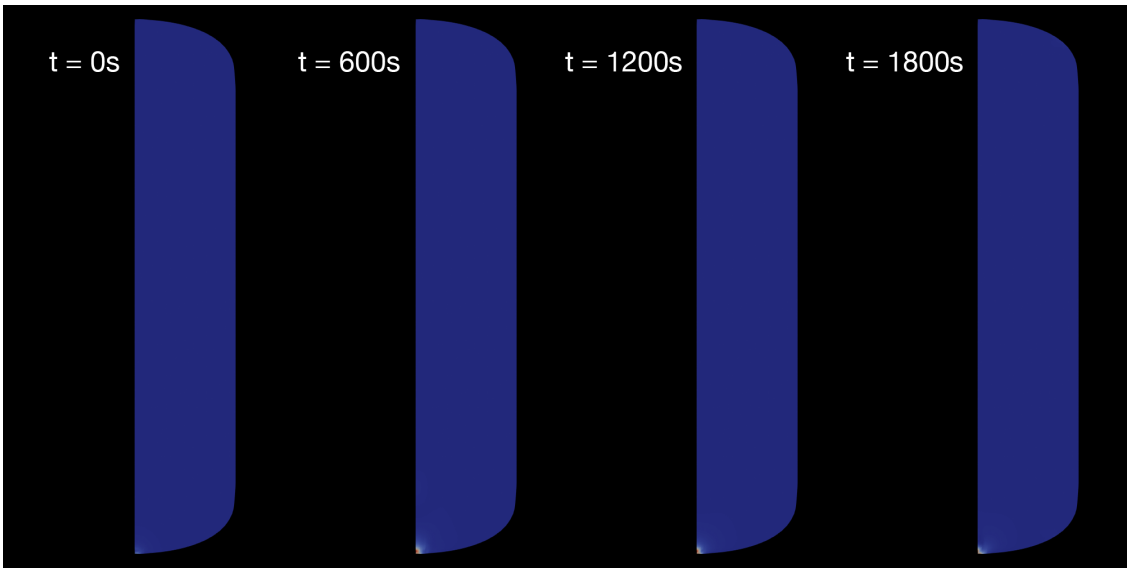


Figure 5.16.: Average temperature of the fluid inside the tank over time



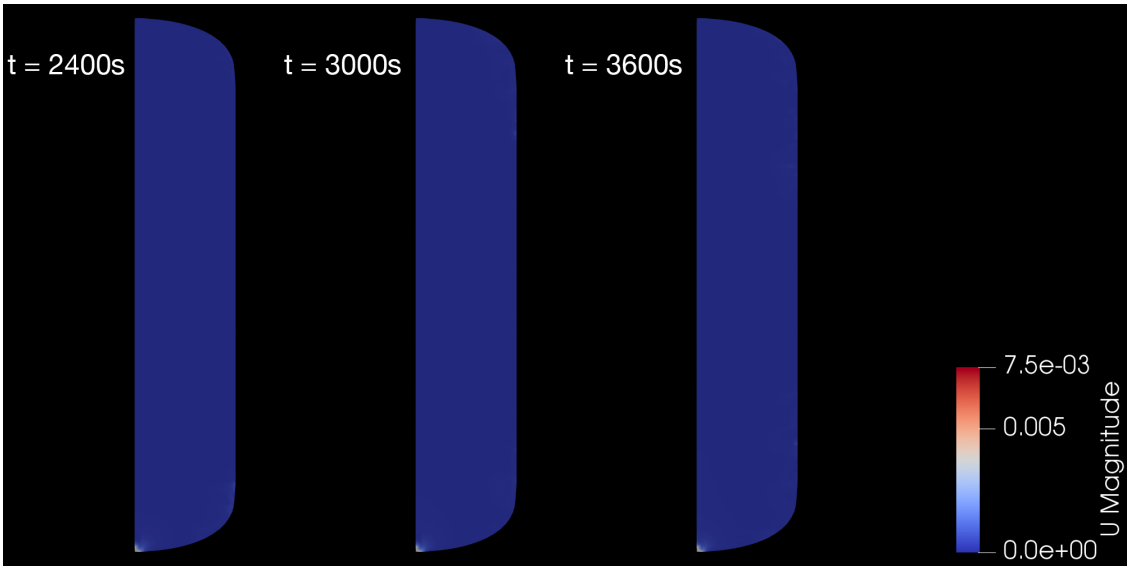


Figure 5.17.: Fluid velocity $[\frac{\mathrm{m}}{\mathrm{s}}]$ inside the tank at different times

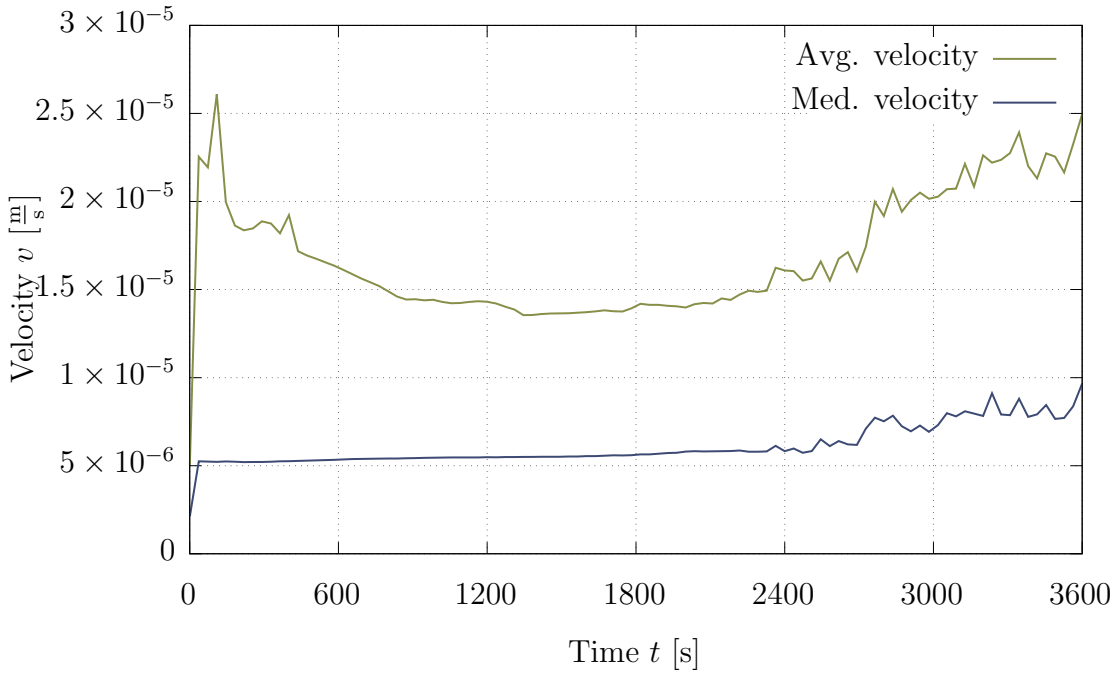


Figure 5.18.: Average velocity magnitude of the fluid inside the tank over time

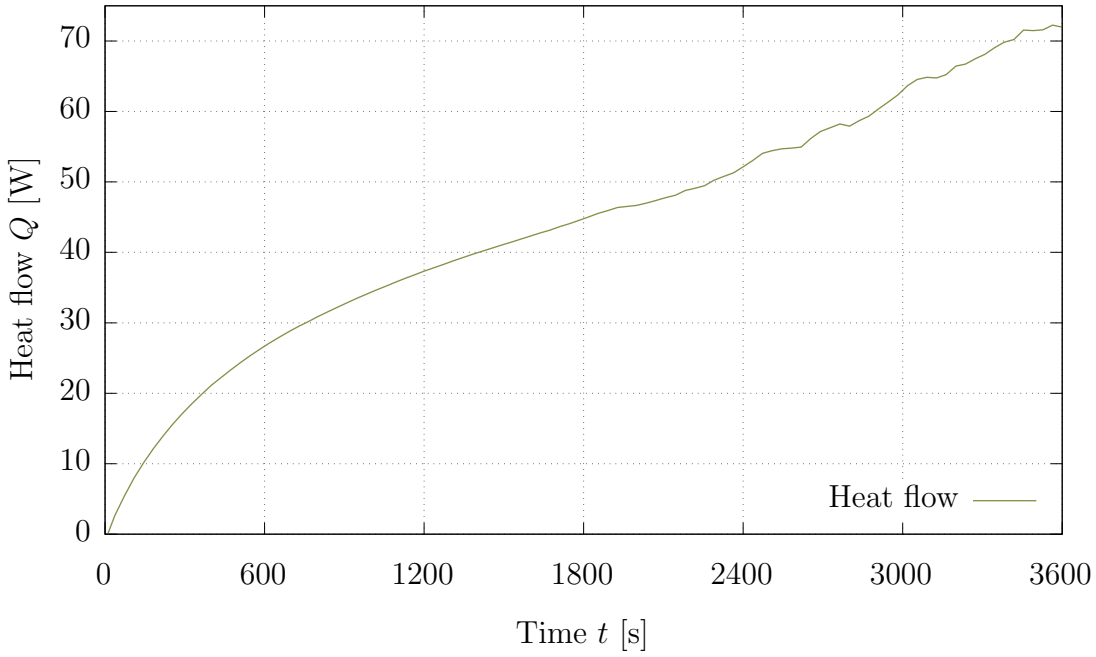


Figure 5.19.: Average velocity magnitude of the fluid inside the tank over time

5.3.1. EcosimPro comparison

The EcosimPro model described in chapter 4.5.2 was slightly modified in order to apply it to the servicer case. The results for the pressure and temperature decrease of the EcosimPro model and the OpenFOAM model are compared in Fig. 5.20 and Fig. 5.21. The EcosimPro model predicts a slower pressure decrease compared to OpenFOAM and a near-constant fluid temperature. This is not the expected behavior, as the depressurization results in expansion work and thus energy is being transferred out of the tank. This decreases the propellant temperature. A possible reason for this modeling error is the zero dimensionality of the tank model used by EcosimPro. This assumes a uniform distribution of all fluid properties inside the tank volume and might result in an overestimation of the heat transfer over the tank wall, resulting in a smaller temperature drop. As is the client case, the boundary condition used in the OpenFOAM model might also be difficult to transfer to a 1D/0D simulation again.

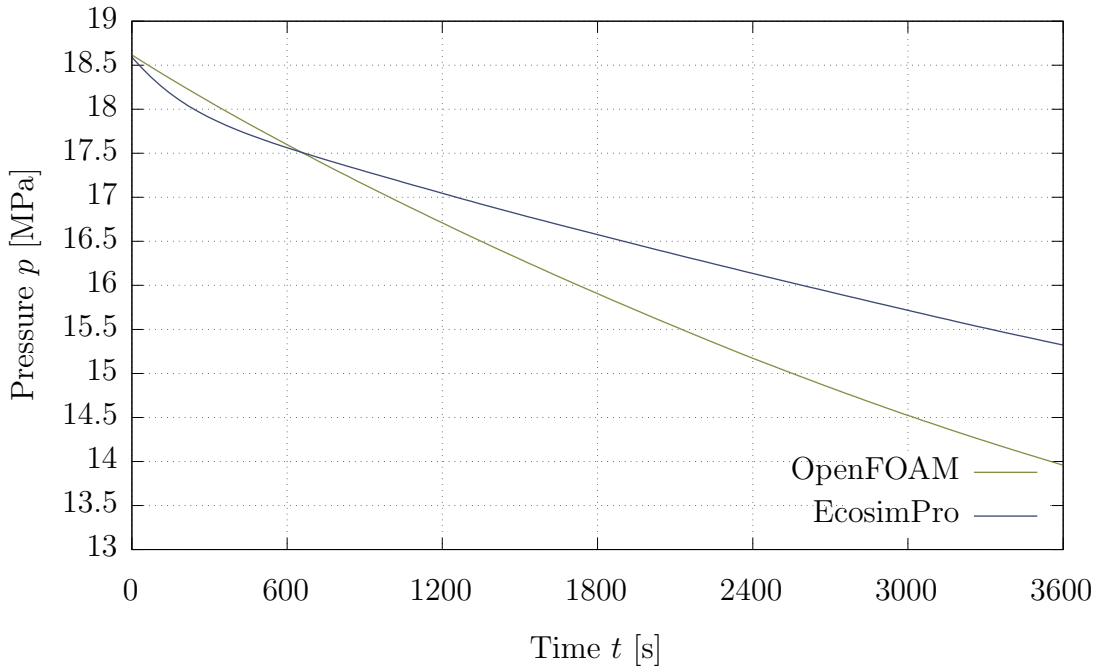


Figure 5.20.: Fluid pressure according to the OpenFOAM and the EcosimPro simulation over time

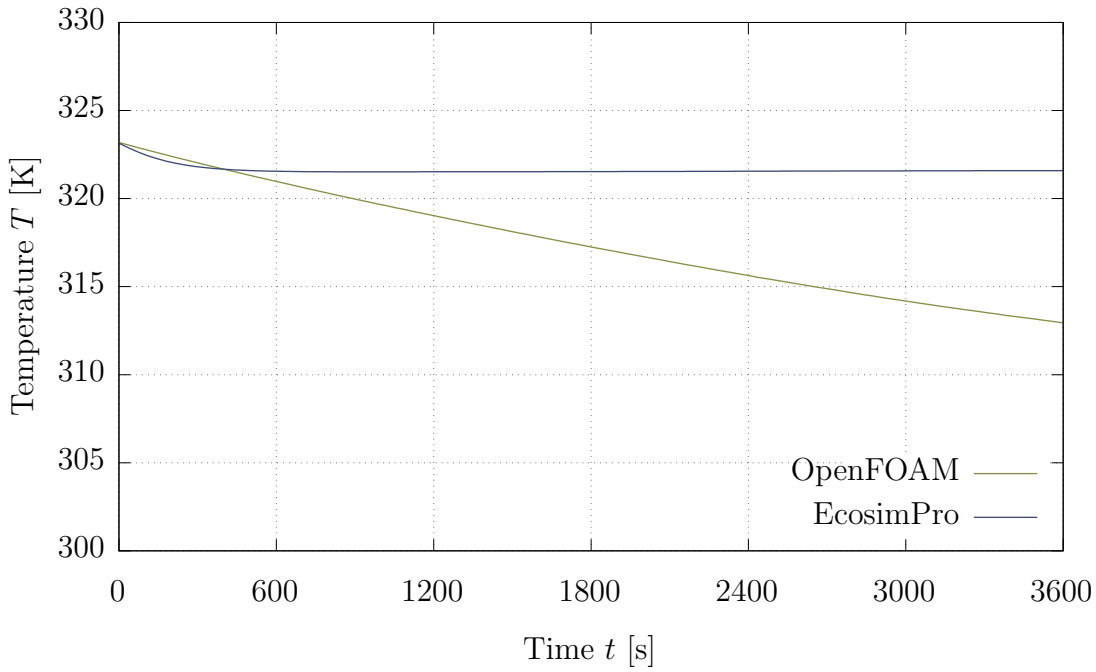


Figure 5.21.: Average fluid temperature according to the OpenFOAM and the EcosimPro simulation over time

6. Conclusion

6.1. Summary

Within the scope if this thesis, a two-dimensional model of xenon inside a tank during the loading and unloading process in a zero-g environment has been created. First, the background to the problem in form of the *ESPRIT* has been explained and two alternative modeling tools have been presented that were later used to built comparison models. Subsequently, the relevant theoretical aspects of the fluid flow description have been summarized, including the derivation of the conservation equations for the momentum and the energy, as well as the introductions of the relevant dimensionless quantities. Following this, the OpenFOAM solver selection process has been detailed and the algorithm behind the chosen solver, *rhoPimpleFoam*, has been presented. Additionally, some important fundamentals of CFD modeling, namely the temporal and convection discretization schemes, and the turbulence modeling, have been described.

The two-dimensional tank model created using OpenFOAM has been presented in detail. Initially, a general OpenFOAM model setup has been described. The chosen mesh topology of a wedge shape as well as the splitting procedure for a multithreaded simulation have been explained. Following this, the numerical schemes and solution algorithms selected for the model have been presented and discussed. The method used to obtain the fluid properties during the simulation from two-dimensional value tables extracted from the NIST Chemistry WebBook as well as the selected turbulence modeling settings have been described. Subsequently, the chosen initial and boundary conditions have been discussed. A velocity driven flow was chosen with the inlet/outlet velocity calculated from a fixed mass flow and the density at each time step. The fluid temperature at the wall was fixed to a given value using a Dirichlet boundary condition. Furthermore, the two comparison models have been presented: Firstly, an Excel tool using a 0D approach to calculate the wall temperature of the tank, the temperature of the fluid inside the tank, and the heat flow from the tank wall to the ambient air. Secondly, an EcosimPro model created to mirror the boundary and initial conditions selected for the OpenFOAM model as closely as possible.

Finally, the results of the OpenFOAM calculations have been discussed. Initially, the results of a number of parameter studies that were used to find optimal values for the cell size, the time step and the number of processor cores have been presented. Using these values, results for the client case were calculated and have been described here. The chosen settings have been selected with a focus on the numerical stability as a stable model was one of the prime requirements. The temperature has been shown to rise during the pressurization as expected. A value for the required heat flow out of the tank to generate the prescribed fluid temperature at the wall has been shown to rise to roughly 70 W during the first hour of the pressurization process. The results have been compared to the EcosimPro model described previously. The EcosimPro predictions of the pressure increase have been proven to match the 2D simulation closely, but the temperature increase is underestimated. Possible reasons for this have been discussed. The Excel tool could not be used for a direct comparison, but a qualitative comparison of the calculated heat flow has been performed. For the servicer case, the results have been discussed as well. The heat flow into the tank has been calculated and presented as well. As with the client, the results of the OpenFOAM servicer calculation have been compared to an EcosimPro calculation as well. Here, a larger discrepancy between the results has been observed, especially for the average fluid temperature. Again, possible reasons for the discrepancy have been discussed.

As expected, the thermal loads on the tank during the loading and unloading process are relatively large. To keep a constant fluid temperature at the tank wall, roughly 70 W of heat is required to be extracted from or inserted into the fluid at one hour of the pressurization or depressurization process with a flow rate of $3\,\frac{\mathrm{g}}{\mathrm{s}}$. If a higher flow rate is desired, e.g. to speed up the fueling process, the required heat flow would increase even further. Thus, it is vital to consider the loading and unloading procedures when designing the thermal control for the propulsion system. The tank model created as part of this thesis is stable and can be used to analyze the behavior of xenon inside the tank. The 2D simulation has been shown to model behavior that is not modeled by EcosimPro, making it a valuable addition to the tools available during the design of the thermal and propulsion system of a spacecraft. A challenge for the usability of the 2D model is the relatively long computation time, especially compared to the 0D/1D approaches presented above. This makes repeated calculations, for example during a design parameter study particularly cumbersome and presented a difficult hurdle during the model creation.

6.2. Future Work

There are at least two other cases of interest in regards to the behavior of xenon inside a tank. The first is the on-ground fueling case. Here, xenon is loaded into the tank which is already mounted inside a spacecraft. To keep the tank inside its specified temperature range, the tank wall has to be cooled. This happens passively through thermal radiation and natural convection between the outside tank wall and the surrounding air. Additionally, a forced convection can be created, for example by blowing air around the tank during the loading process. This method was evaluated for NASA's Asteroid Redirect Robotic Mission and was found to reduce the tank wall temperature significantly.[13] The second case of interest is the thermal propellant gauging in orbit. Here, the goal is to determine the amount of remaining propellant inside a tank. To achieve this, the tank wall is locally heated and the reaction of the fluid is measured using temperature elements on the outside of the tank. By analyzing the heating curves, conclusions about the remaining propellant level can be drawn.

For both these cases, the tank has to be extended to include the tank wall as well as the surrounding fluid for the on-ground fueling case. This can be achieved by coupling a CFD simulation as performed by OpenFOAM with a finite element simulation of the tank wall. Alternatively, the OpenFOAM solver *chtMultiRegionFoam* could be used. This solver is based on the PIMPLE algorithms, just like the *rhoPimpleFoam* solver utilized in this thesis. Thus, it should be possible to reuse major parts of the model presented in this thesis. The *chtMultiRegionFoam* solver is able to model one or more compressible fluids as well as solid regions. Therefore, a combined model can be created for the propellant fluid inside the tank, the fluid surrounding the tank and the tank itself using OpenFOAM. For the gauging case, there would be no surrounding fluid and the heating elements could be modeled using a localized boundary condition on the outside of the tank wall.

Due to the increased complexity of this new model, it might be necessary to perform these calculations on a more powerful machine, for example a computing cluster. Alternatively, other methods to reduce the computational effort could be used, like increasing the mesh coarseness or relaxing the allowed residuals. However, these measures will have a negative effect on the model's stability and precision.

List of Figures

1.1. Planned configuration of the *Gateway* with the *ESPRIT* module **B** [31] . . . 3
1.2. Picture and schematics of the 80458-1 [34] . . . 4

2.1. **a** Eulerian and **b** Lagrangian representation of the fluid flow . . . 7
2.2. Flow regimes over a flat plate for different Reynolds number values [26] . . . 13
2.3. Thermal and hydrodynamic boundary layer for **a** $Pr < 1$ and **b** $Pr > 1$ [26] . . . 14

3.1. Xenon enthalpy H over the pressure p for different temperatures T. Red line represents critical pressure [23] . . . 17
3.2. Flow chart of the SIMPLE algorithm . . . 20
3.3. Flow chart of the PISO algorithm for each time step . . . 22
3.4. Simplified flow chart of the PIMPLE algorithm . . . 23
3.5. 1D profile of the Upwind scheme . . . 27
3.6. 1D profile of the Central Difference scheme . . . 28
3.7. 1D profile of the Second Order Upwind scheme . . . 29

4.1. Overview of the OpenFOAM case structure . . . 33
4.2. Example of wedge shape . . . 34
4.3. Schematics of the EcosimPro model . . . 43

5.1. Calculation time and continuity error for different mesh cell lengths 46
5.2. Calculation time and continuity error for different time steps . . . 47
5.3. Calculation time for different number of computing threads . . . 48
5.4. Fluid pressure [Pa] inside the tank at different times . . . 50
5.5. Pressure of the fluid inside the tank over time . . . 51
5.6. Average temperature of the fluid inside the tank over time . . . 51
5.7. Fluid temperature [K] inside the tank at different times . . . 52
5.8. Fluid velocity $[\frac{\mathrm{m}}{\mathrm{s}}]$ inside the tank at different times . . . 53
5.9. Average velocity magnitude of the fluid inside the tank over time . 54
5.10. Heat flow out of fluid through the tank wall over time . . . 54

5.11. Fluid pressure according to the OpenFOAM and the EcosimPro simulation over time 55
5.12. Average fluid temperature according to the OpenFOAM and the EcosimPro simulation over time 56
5.13. Fluid pressure [Pa] inside the tank at different times 57
5.14. Pressure of the fluid inside the tank over time 58
5.15. Fluid temperature [K] inside the tank at different times 59
5.16. Average temperature of the fluid inside the tank over time 60
5.17. Fluid velocity $[\frac{m}{s}]$ inside the tank at different times 61
5.18. Average velocity magnitude of the fluid inside the tank over time . 61
5.19. Average velocity magnitude of the fluid inside the tank over time . 62
5.20. Fluid pressure according to the OpenFOAM and the EcosimPro simulation over time 63
5.21. Average fluid temperature according to the OpenFOAM and the EcosimPro simulation over time 63

List of Tables

3.1. Typical values for the $k - \varepsilon$ model constants[9] 31

4.1. Initial conditions inside the tank for the client and servicer case . . 37
4.2. Inlet/outlet port and wall boundary conditions used for turbulent flow properties 40

5.1. Number of mesh cells for different cell sizes 45

Bibliography

[1] Abdallah, S. and Dreyer, J. "Dirichlet and Neumann boundary conditions for the pressure poisson equation of incompressible flow". In: *International Journal for Numerical Methods in Fluids* 8.9 (1988), pp. 1029–1036. DOI: 10.1002/fld.1650080905.

[2] Ambatipudi, Vaidehi. *SIMPLE Solver for Driven Cavity Flow Problem*. 2010.

[3] Anderson, John D. *Computational Fluid Dynamics: The Basics with Applications*. McGraw-Hill Inc., 1995. ISBN: 9780070016859.

[4] Behrens, Tim. *OpenFOAM's basic solvers for linear systems of equations*. 2009.

[5] Bergman, T.L. et al. *Fundamentals of Heat and Mass Transfer*. Wiley, 2017. ISBN: 9781119337676.

[6] Bondarev, Alexander E. and Kuvshinnikov, Artem E. "Analysis of the Accuracy of OpenFOAM Solvers for the Problem of Supersonic Flow Around a Cone". In: *Computational Science – ICCS 2018*. Cham: Springer International Publishing, 2018, pp. 221–230. DOI: 10.1007/978-3-319-93713-7_18.

[7] Bravais, Patrick and Grassin, T. *Xenon Ground Support Equipment for Plasmic Propulsion System*. 1999.

[8] EA Internacional. *EcosimPro, System Modelling and Simulation Software*. URL: https://www.ecosimpro.com/products/ecosimpro/ (visited on 20/11/2019).

[9] Ferziger, Joel H. and Perić, Milovan. *Computational Methods for Fluid Dynamics*. 2002. DOI: 10.1007/978-3-642-56026-2.

[10] Forrester, Chris. *Beyond Frontiers*. Broadgate Publications, Sept. 2016.

[11] Ganapathi, Gani B and Engelbrecht, Carl S. "Performance of the xenon feed system on Deep Space One". In: *Journal of Spacecraft and Rockets* 37.3 (2000), pp. 392–398.

[12] Gilligan, Patrick and Tomsik, Thomas. *Modeling ARRM Xenon Tank Pressurization Using 1D Thermodynamic and Heat Transfer Equations*. Aug. 2016. URL: https://ntrs.nasa.gov/search.jsp?R=20170003928.

[13] Gilligan, Ryan P. and Tomsik, Thomas M. *Modeling Xenon Tank Pressurization using One-Dimensional Thermodynamic and Heat Transfer Equations*. Apr. 2017. URL: https://ntrs.nasa.gov/search.jsp?R=20170004518.

[14] GNU Project. *GNU General Public License v3.0*. URL: https://www.gnu.org/licenses/gpl-3.0.en.html (visited on 15/10/2019).

[15] Hellingman, C. "Newton's third law revisited". In: *Physics Education* 27.2 (Mar. 1992), pp. 112–115. DOI: 10.1088/0031-9120/27/2/011.

[16] Hirsch, C. *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics*. 2nd ed. Butterworth-Heinemann, 2007. ISBN: 9780750665940.

[17] Holzmann, Tobias. *Mathematics, Numerics, Derivations and OpenFOAM®*. 2018. DOI: 10.13140/RG.2.2.27193.36960.

[18] Intel Corporation. *Xeon® Silver 4114 Processor Data Sheet*. URL: https://ark.intel.com/content/www/us/en/ark/products/123550.html (visited on 15/11/2019).

[19] Issa, R. I. "Solution of the implicitly discretised fluid flow equations by operator-splitting". In: *Journal of Computational Physics* 62 (Jan. 1986), pp. 40–65. DOI: 10.1016/0021-9991(86)90099-9.

[20] ITP Engines UK Ltd. *ESATAN-TMS Products Overview*. URL: https://www.esatan-tms.com/products/product.php (visited on 15/11/2019).

[21] Kolmogorov, Andrei Nikolaevich. "Dissipation of energy in the locally isotropic turbulence". In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 434.1890 (1991), pp. 15–17. DOI: 10.1098/rspa.1991.0076.

[22] Kugelberg, Joakim et al. "Accommodating electric propulsion on SMART-1". In: *Acta Astronautica* 55.2 (2004), pp. 121–130. DOI: 10.1016/j.actaastro.2004.04.003.

[23] Linstrom, P.J. and Mallard, W.G. *NIST Chemistry WebBook, NIST Standard Reference Database Number 69*. DOI: 10.18434/T4D303.

[24] Liu, Fangqing. "A Thorough Description Of How Wall Functions Are Implemented In OpenFOAM". In: *Proceedings of CFD with OpenSource Software* (2016). URL: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2016/FangqingLiu/openfoamFinal.pdf.

[25] Moral, J et al. "ESPSS Simulation Platform". In: May 2010.

[26] Moukalled, Fadl, Mangani, Luca and Darwish, Marwan. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab®*. 2015. ISBN: 978-3-319-16873-9. DOI: 10.1007/978-3-319-16874-6.

[27] Mukha, Timofey and Liefvendahl, Mattias. "Large-Eddy Simulation of Turbulent Channel Flow". In: 2015.

[28] Münzberg, H.G. *Flugantriebe: Grundlagen, Systematik und Technik der Luft- und Raumfahrtantriebe*. Springer Berlin Heidelberg, 1972. ISBN: 9783662117583.

[29] Murthy, Jayathi Y. *Numerical Methods in Heat, Mass, and Momentum Transfer*. 2002.

[30] NASA. *Humanity's Return to the Moon*. URL: https://www.nasa.gov/specials/artemis (visited on 15/11/2019).

[31] NASA. *NASA's New Spaceship*. Nov. 2018. URL: https://www.nasa.gov/feature/questions-nasas-new-spaceship (visited on 15/11/2019).

[32] Nguyen, Hugo, Köhler, Johan and Stenmark, Lars. "The merits of cold gas micropropulsion in state-of-the-art space missions". In: *IAF abstracts, 34th COSPAR Scientific Assembly*. American Institute of Aeronautics and Astronautics, Jan. 2002, p. 785.

[33] Norris, Stuart Edward. *A Parallel Navier Stokes Solver for Natural Convection and Free Surface Flow*. University of Sydney. Engineering, 1980. URL: http://hdl.handle.net/2123/376.

[34] Northrop Grumman. *80458-1 Data Sheet*. Tech. rep. June 2018. URL: https://www.northropgrumman.com/Capabilities/PressurantTanks/Documents/DS458.pdf.

[35] OHB SE. *Press release: OHB participating in the ESPRIT module for future lunar orbital Gateway*. Sept. 2018. URL: https://www.ohb.de/en/news/ohb-participating-in-the-esprit-module-for-future-lunar-orbital-gateway/ (visited on 15/11/2019).

[36] OHB System AG. *SmallGEO - The multi-purpose geostationary satellite platform*. Apr. 2018. URL: https://www.ohb.de/fileadmin/ohb/Downloads/OHB-System_SmallGEO_platform_2018.pdf.

[37] OpenCFD Ltd. *OpenFOAM launched 10th December 2004*. URL: https://web.archive.org/web/20050208124617/http://www.opencfd.co.uk/openfoam/launch.html (visited on 01/11/2019).

[38] Patankar, S.V and Spalding, D.B. "A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows". In: *International Journal of Heat and Mass Transfer* 15.10 (1972), pp. 1787–1806. ISSN: 0017-9310. DOI: 10.1016/0017-9310(72)90054-3.

[39] Schlichting, Hermann and Gersten, Klaus. *Boundary-Layer Theory*. 9th ed. Springer Berlin Heidelberg, 2017. DOI: 10.1007/978-3-662-52919-5.

[40] Shafranovich, Y. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. RFC 4180. Network Working Group - IETF, Oct. 2005. URL: https://tools.ietf.org/html/rfc4180.

[41] Showalter, Ralph E. *Hilbert Space Methods in Partial Differential Equations*. Pitman Publishing, 1979. ISBN: 9780273084402.

[42] Shu, Jian-Jun, Bin Melvin Teo, Ji and Kong Chan, Weng. "Fluid Velocity Slip and Temperature Jump at a Solid Surface". In: *Applied Mechanics Reviews* 69.2 (Mar. 2017). DOI: 10.1115/1.4036191.

[43] SimScale. *CFD: PIMPLE Algorithm*. URL: https://www.simscale.com/forum/t/cfd-pimple-algorithm/ (visited on 21/10/2019).

[44] SimScale. *Turbulent Pipe Flow*. URL: https://www.simscale.com/docs/content/validation/TurbulentPipeFlow/TurbulentPipeFlow.html (visited on 01/11/2019).

[45] SpaceX. *Starlink Mission Press Kit*. May 2019. URL: https://www.spacex.com/sites/spacex/files/starlink_press_kit.pdf.

[46] The OpenFOAM Foundation. *OpenFOAM v6 User Guide*. URL: https://cfd.direct/openfoam/user-guide-v6 (visited on 01/11/2019).

[47] White, F.M. *Heat transfer*. Addison-Wesley Longman, 1984. ISBN: 9780201083248.

[48] Yuusha. *tabulatedThermophysicalProperties*. URL: https://github.com/Yuusha0/tabulatedThermophysicalProperties (visited on 15/10/2019).

A. Usage guide for Python scripts

Two of the Python scripts written during the course of this work can be useful for other, similar projects. Their usage will be explained here.

A.1. nistToOpenFoam.py

This script can be used to create the two dimensional lookup tables that are required by the OpenFOAM extension `tabulatedThermophysicalProperties`. In addition to converting the data from isothermal and isobaric CSV files into the 2D tables, it can also obtain the files directly from the NIST Chemistry WebBook before converting them.

To simply convert already downloaded files, sort the files in two subfolders called `isothermal` and `isobaric` and run

```
$ nistToOpenFoam.py DIR
```

where `DIR` is the path to the directory containing the two subfolders and the location where the newly created 2D tables will be stored.

To download the data from NIST directly, run

```
$ nistToOpenFoam.py -d ID TLOW THIGH TSTEP PLOW PHIGH PSTEP DIR
```

where `DIR` is the path to the directory in which to store the files again, `TLOW` and `THIGH` define the temperature range for which to get the data with a temperature step of `TSTEP`. All those values have to be given in Kelvin. `PLOW`, `PHIGH`, and `PSTEP` have the same functionality for the pressure range, with the values given in MPa. The `ID` is the fluid id used by NIST. This can be extracted from the URL of the *Fluid Properties* page after searching for a fluid on the website. For xenon, the id is `C7440633`. For the download functionality to work, the Python environment must have access to the NIST website. This means that a proxy setting might be necessary. To set a proxy on Windows run

```
$ set http_proxy=http://proxyurl:PORT
$ set https_proxy=http://proxyurl:PORT
```

with the `proxyurl` replaces by the URL of the proxy and the `PORT` by the proxy's port number.

A.2. meshGen.py

The `meshGen.py` script is used to generate a `blockMeshDict` file, which is used by OpenFOAM's *blockMesh* utility to create the mesh in the format required by OpenFOAM. The `meshGen.py` script in its current form is specific to a tank shape, but it could easily be adapted to different geometries.

To specify the diameters of the tank as well as the mesh size steps, a file called `meshGen.ini` has to be present in the `system` directory of an OpenFOAM model. The `meshGen.ini` has to be of the form

```
[geometry]
# Angle of the wedge [°]
wedgeAngle = 0.1
# Radius of the cylindrical part [mm]
rCyl = 210
# Heigt of cylindrical part [mm]
hCyl = 830
# Height of curved dome [mm]
hDome = 140
# Radius at which the dome curve begins [mm]
rDome = 20
# Radius of port [mm]
rPort = 10

[mesh]
# Size of the cells in x direction
deltaX = 5

# Size of the cells in z direction
deltaZ = 5
```

with the values adapted to the specific case. Lines starting with a `#` are comment lines and will be ignored by the script. The header blocks with the square brackets (e.g `[geometry]`) are required however. With the `meshGen.ini` file in place, the `blockMeshDict` can be generated simply by calling

```
$ meshGen.py
```

inside an OpenFOAM case.